TR-I-0170

音声認識用言語モデル構築に関する考察

Design Principle of Language Model for Speech Recognition

坂野俊哉　　森元逞

Toshiya Sakano　Tsuyoshi Morimoto

1990.8

概要

In speech recognition, the recognition rate can be improved by using statistical information about linguistic texts. However, there is no criterion for selecting sample linguistic texts from those available. If unbalanced linguistic text samples are selected, the information extracted would not be suitable for a linguistic model. To solve this problem, we need to quantitatively analyze linguistic texts.

In this report, we introduce a method to quantitatively analyze linguistic texts, and propose a method for selecting linguistic texts. Moreover, we describe the possibility of developing a criterion for selecting test texts needed for system evaluation, and show the relationship between recognition system performance and the features of the sample text group used for the linguistic model.

ＡＴＲ自動翻訳電話研究所

ＡＴＲ Interpreting Telephony Research Laboratories

もくじ

1 Introduction

Modeling is a significant element of speech recognition. Language models are necessary in order to achieve effective recognition, for example, a stochastic grammar which can parse sentences effectively. Of course, a language model is established from linguistic data such as conversation texts. Because it is necessary that linguistic data be parsed, and their grammatical structures determined, it is very difficult to gather sufficient linguistic data to reflect all natural language features. Accordingly, until now, as much linguistic data as possible has been gathered to establish language models with regard to whether the linguistic data are necessary or not. Such linguistic data are very contrived, that is, not natural. When the gathered linguistic data are unbalanced with respect to unnecessary features, the language model made from them may also be unbalanced with respect to the unnecessary features. If a language model is to be made with respect to grammatical features, the unbalance with respect to unnecessary features must be removed. Therefore, sample data to make a language model with respect to grammatical features should be selected uniformly with respect to unnecessary features. Similarly, in order to test the language model uniformly, the linguistic test data should be selected uniformly with respect to grammatical features.

This paper propose a metric between linguistic texts, and describes the characteristics of a text space derived from the metric by using quantification theory IV.

2 Quantification Theory IV

The quantification theory is one of the quantification methods proposed by Chikio Hayashi. There are four quantification theories, and quantification theory IV is the fourth one. Quantification theory IV can quantify data which have no numerical value. For example, because linguistic texts are not numerical, we adopted the quantification theory to quantify the linguistic texts.

For this procedure, similarity among data must be defined. By this method, data whose similarity values are approximately the same are quantified closely, and data whose similarity values are very different are quantified distantly. Mathematically, quantification theory IV is used to calculate the eigenvalues of a certain symmetric matrix and their eigenvectors. The matrix is derived from data similarity.

3 Metric and Similarity of Conversation Texts

3.1 Eample 1: *Topical Metric and Similarity*

Conversation texts include nouns, and the patterns of noun co-occorrence are all different. We think that the pattern of noun co-occorrence in a conversation text reflects its topics. The distribution of noun occurrence in a conversation text can be regarded as its pattern of noun co-occurrence.

Let t_i and t_j be conversation texts and let P_i and P_j be the probability distribution functions for the nouns in t_j and t_j respectively. For example, $P_i(ns)$ denotes the probability that noun ns exists in the conversation text t_i. Then, the metric d_w between t_i and t_j is defined as follows:

$$d_w(t_i, t_j) = \sqrt{\sum_{w \in t_i \cup t_j} \{P_i(w) - P_j(w)\}^2}$$

If t_i is the same conversation text as t_j, $d_w(t_i, t_j)$ is 0. However, the converse is not true. The value range of $d_w(t_i, t_j)$ is from 0 to $\sqrt{2}$(this is proved very easily).

Now we can define the noun pattern similarity e_w as follows:

$$e_w(t_i, t_j) = -d_w(t_i, t_j)$$

3.2 Example 2: *Grammatical Metric and Similarity*

Each sentence in a linguistic text is able to be parsed using a linguistic grammar. Usually, the parsed result is a sequence of rewriting grammar rules. That is, a parsed result of a sentence reflects its pure grammatical structure. Therefore, a conversation text is regarded as a set consisting of sequences of rewriting grammar rules by a linguistic grammar.

In the same manner as noun pattern similarity, a grammatical pattern similarity can be defined. Let t_i and t_j be conversation texts and let P_i and P_j be the probability distribution functions for the parsed results in t_j and t_j respectively. Then, the metric d_g between t_i and t_j is defined as follows:

$$d_g(t_i, t_j) = \sqrt{\sum_{g \in G} \{P_i(g) - P_j(g)\}^2}$$

(G is a treated grammar and g means a grammar pattern defined by the grammar G.)
Now we can define the grammatical pattern similarity e_g as follows:

$$e_g(t_i, t_j) \;=\; -d_g(t_i, t_j)$$

4 Sample Data for a Language Model

4.1 Filtering Out Unnecessary Elements from Sample Data

When statistical features are introduced into a linguistic grammar such as a stochastic grammar, they must be established by a linguistic corpus. Then, as many natural linguistic data as possible are necessary. The words "natural linguistic data" mean that the data is not unbalanced with respect to unnecessary elements other than grammatical properties. For example, generally, topics have nothing to do with such a grammatical elements. If the linguistic data is unbalanced with respect to properties which are not grammatical properties, the established stochastic grammar is effective to the superior elements which have nothing to do with grammatical properties of liguistic texts, but not effective to the inferior elements. Because we should make a stochastic grammar reflecting the pure grammatical properties, it is needed to reduce as many unnecessary properties as possible for the stochastic grammar.

For this purpose, we define a metric and a similarity between the texts with respect to unnecessary elements other than grammatical properties, and quantify the texts by using quantification theory IV with the metric and the similarity to represent the texts in the n-dimensional text space. In order to filter out the unnecessary elements, the mesh method can be used to select the sample texts from the n-dimensional text space with respect to the unnecessary elements. By this way, sample texts uniform with respect to unnecessary elements other than grammatical properties can be selected from linguistic text database. The details are explained in the following sections.

4.2 Mesh Method

Generally, conversation texts quantificated by using quantification theory IV are represented as points in an n-dimensional Euclidean space. We call this space n-dimensional text space. In order to uniformly select the sample texts from the text space, a mesh method is usefull. In this method, first, the text space is represented by cubes(called meshes) defined as a direct product of each axis separated into equally long segments, and texts of the same number are selected from each mesh. The number of selected texts depends on the size of the mesh and the number of texts selected from one mesh. However, the number of texts selected from one mesh depends on the size of the mesh. If the mesh is very small, there are some meshes with no texts. Therefore, the size of the mesh and the number of texts selected from one mesh should be determined prudently.

4.3 Leaning Degree

Definition 1 *Let $X = \{x_i\}_{0 \le i \le r}$ be a real number sequence such that*

$$x_0 \le x_1 \le \cdots \le x_r$$

Let $D = \{d_i\}_{1 \le i \le r}$ be the intervals set such that

$$d_1 = x_1 - x_0,\ d_2 = x_2 - x_1, \cdots,\ d_r = x_r - x_{r-1}$$

If $x_0 \neq x_r$, we define the leaning degree L of X as follows,

$$L = \frac{r\sqrt{V[D]}}{x_r - x_0} \quad (V[D] \text{ means the variance value of } D)$$

In this definition, if all numbers in X line up at regular intervals, the leaning degree of X is 0. The leaning degree L of X means how unbalanced the data in X are, that is, not uniform.

Definition 2 *Let $X = \{x_i\}_i$ be a sequence of points in a n-dimensional Euclidean space, and let $\{a_i\}_i$ be a family of orthonormal axes. Then, the leaning degree L of X is defined as the mean of the leaning degrees for $\{a_i\}_i$. That is,*

$$L = \frac{\sum_{i=1}^{r} L_i}{r} \quad (L_i \text{ is the leaning degree of } a_i.)$$

	Topics					
Texts	A	B	C	D	E	F
data.K010	⊗				⊗	
data.K011	⊗		⊗			
data.K012	⊗	⊗		⊗	⊗	
data.K013		○		○	○	○
data.K014		○		○	○	○
data.K015	○		○			
data.K016	⊗			⊗	⊗	
data.K017	○		○			
data.K018	⊗		⊗			
⋮						
data.T010	○		○		○	
data.T011	○		○			○
data.T012	⊗		⊗	⊗		⊗
data.T013	○		○			
data.T014	○		○			

A:conference B:transportation C:procedure

D:hotel E:information F:sight seeing

表 1: Contents of Texts Files

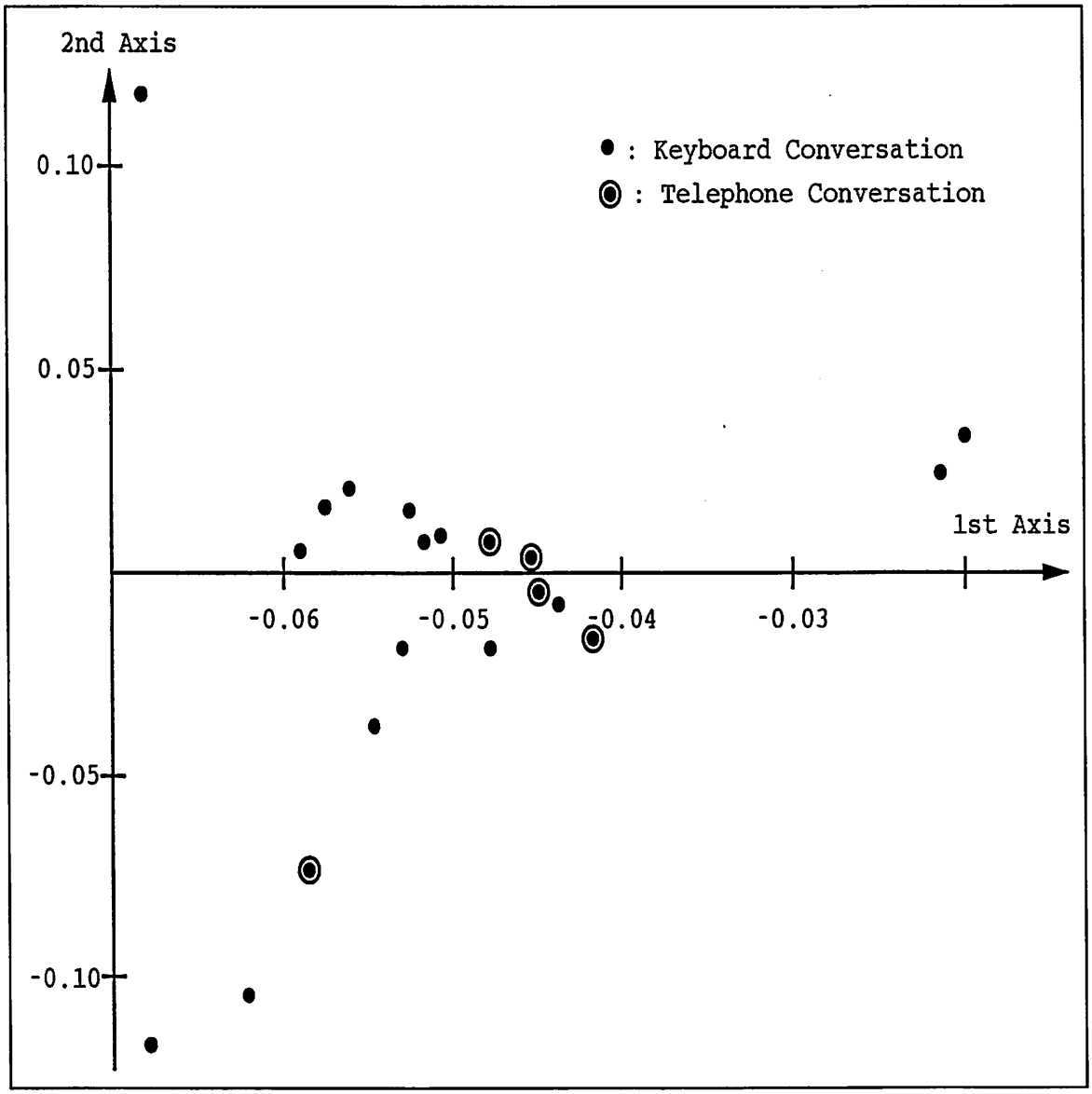

図 1: First and Second Axes of the Text Space

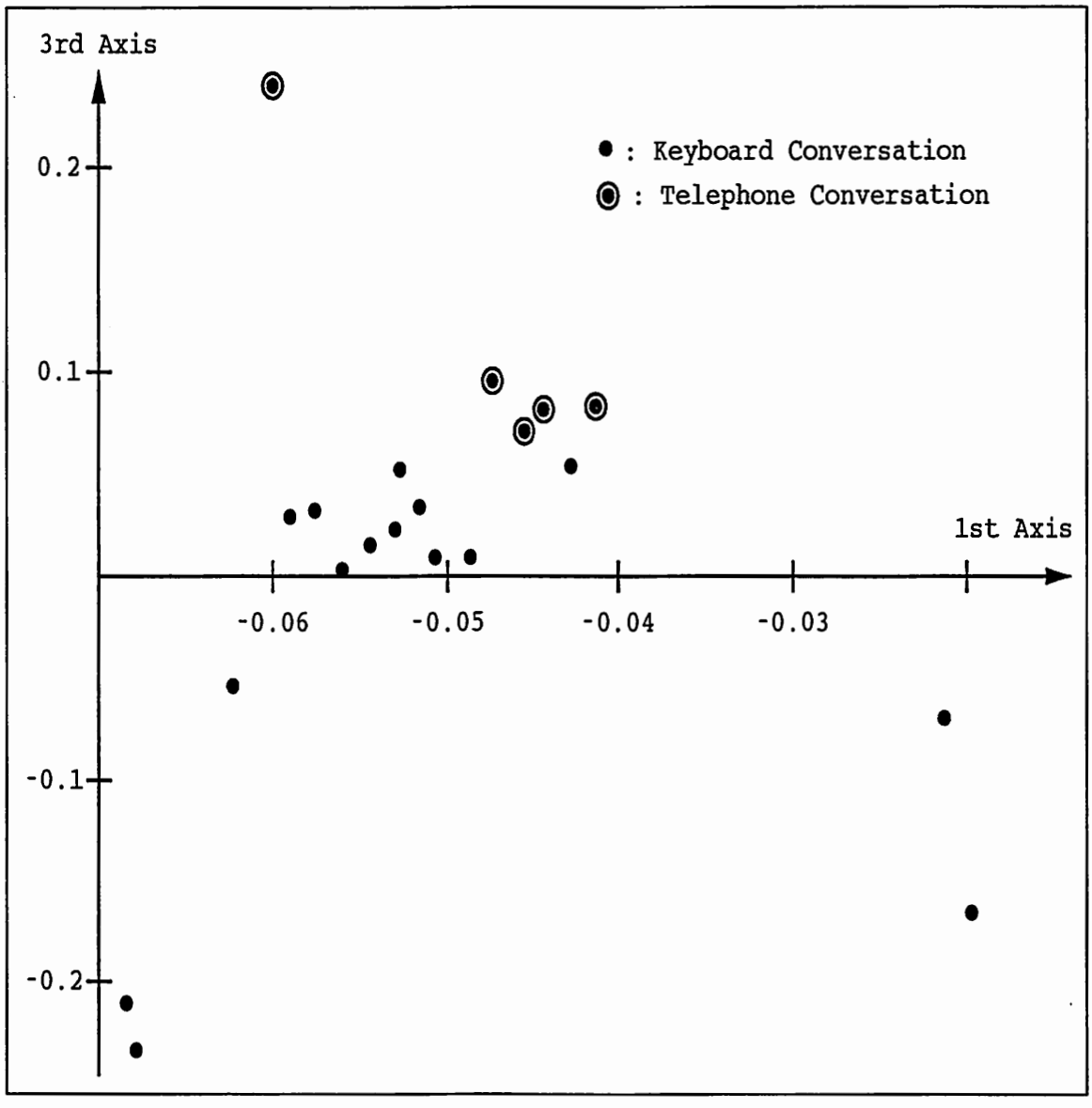

図 2: First and Third Axes of the Text Space

4.4 Example

We experimented with 20 linguistic texts. These linguistic texts are composed of simulated keyboard conversations and telephone conversations . The keyboard conversation texts are stored in 15 files, and the telephone conversations texts in 5. Table 1 shows the texts and their contents as topics. The topics in table 1 are determined by reading them. We analyzed these texts using quantification theory IV with respect to topics of texts by using distributions of nouns in the texts. As a result, the texts can be represented in three-dimensional Euclidean space. Figures 1 and 2 show the space. Judging from the results, the first axis tends to be about sight-seeing or business management, the second axis tends to be about documents or payment, and the third axis tends to be about keyboard conversation or telephone conversation. Last, we selected six files, which are marked in the table 1, from all files by using the mesh method. The leaning degree of all 20 files is 3.032, and the leaning degree of the selected files is 1.462. From table 1, it can be easily understood that these selected files can cover any topic. This shows the effectiveness of the similarity e_w of the conversation text and the mesh method.

5 Density of Text Space

Density of text space is a significant feature. Leaning degree of text space is a statistical feature about the whole of points in the text space. The other side, density of text space is statistical feature about each point in the text space. The relationship between the density of text space with respect to grammar properties effects the performance of a speech recognition system. The definition of density of text space and the relationship between it and the performance of a speech recognition system are described in the following.

5.1 Leaning Density

Definition 3 *Let $X = \{x_i\}_i$ be a sequence of points in a n-dimensional Euclidean space. Then, the leaning density δ_i of x_i is defined as follows:*

$$\delta_i' = \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{1}{d_g(x_i, x_j)^2}$$

$$\delta_i = \frac{\delta_i'}{\sum_{j=1}^{n} \delta_j'}$$

The leaning density δ_i means a measure how many data crowd around x_i.

	Texts	Leaning Density	Recognition Rate
1	data.MC1	0.0035	84 [%]
2	data.MC2	0.0036	81 [%]
3	data.MC3	0.0031	65 [%]
4	data.MC4	0.0043	74 [%]
5	data.MC5	0.0046	90 [%]

表 2: Leaning Density and Recognition Rate of Test Texts

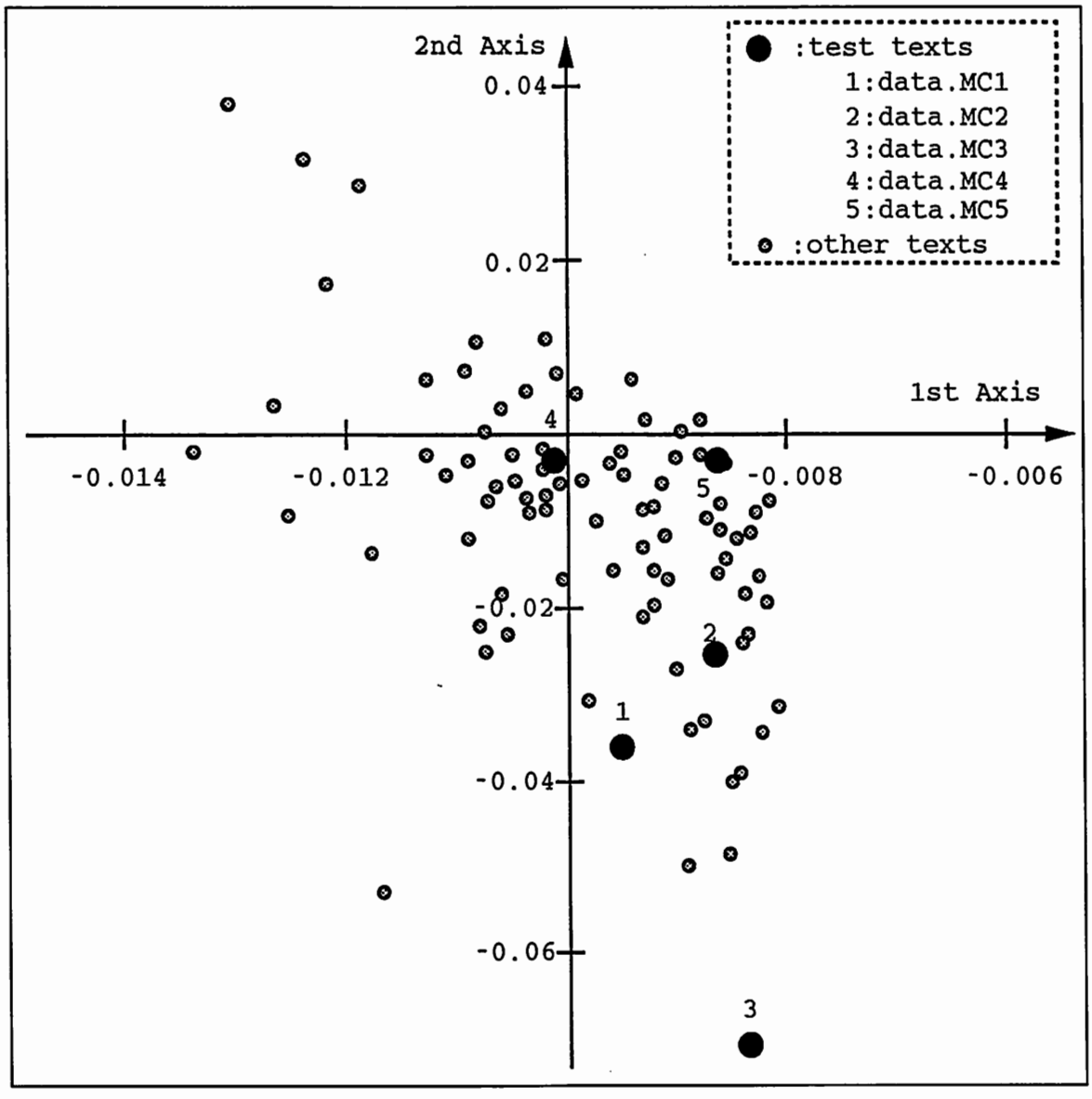

図 3: Grammar Text Space

5.2 Relationship between Leaning Density and System Performance

First, we investigated the text space of 100 sample linguistic texts including the test texts by using quantification theory IV with respect to grammar features. Figure 3 shows the result, and the emphasized points in this figure indicate the test texts. The test texts exist at rather distant points.

Next, we obtained the relationship between leaning density and the performance of a speech recognition system using the test data. Table 2 shows the leaning densities of the five test texts in this text space and the recognition rate. There is a relationship between the leaning densities and the recognition rate of our system(except for 'data.MC4'). That is, the higher the leaning density, the worse the recognition rate of the system becomes.

Judging from these results, leaning density can be used as a measure to presume the recognition rate of a text. Then,the following methods can be possible:

- In the case that some target samples are given for speech recognition, it can improve the performance of the speech recognition system to learn the stochastic grammar of this system with the texts which increase the leaning density of the target samples.

- In the case that a stochastic grammar of a speech recognition system is learned with given sample texts, in order to precisely evaluate the system, the most suitable data(texts) can be selected from text space by mesh method.

6 Conclusion

In this paper, we proposed the metric of linguistic texts, and described that a text space can be derived from a metric, and leaning degree and leaning density can be defined for a measure to presume the recognition rate of a text. Moreover, suitable sample or test texts for modeling or evaluating a speech recognition system can be selected by using mesh method. These will be needed in the future when speech recognition systems become practical. While the experiments we tried are preliminary, in the future, we would like to experiment in detail with more linguistic texts.

7　付録

線形代数演算のためのライブラリのＣ言語ソースコードを付録として掲載する。

7.1　線形代数ライブラリについて

これは、おもにベクトル演算と行列演算を行う関数からなる。概略は次の通りである。

- ベクトル演算
    1. ベクトルの加減乗演算
    2. ベクトル内積演算
    3. ベクトルノルム演算
    4. ベクトル変換
    5. その他ベクトルのコピー・表示
- 行列演算
    1. 行列の加減乗演算
    2. 行列のトレース・行列式
    3. 転置行列
    4. 単位行列
    5. 正方行列の逆行列
    6. 対称行列の固有値・固有ベクトル
    7. その他行列のコピー・表示

ベクトルおよび行列のデータ表現は1次元の実数(double)配列を用いており、各要素の呼び出しはマクロ関数を通じて行う。

1. ベクトル成分呼び出しマクロ $Vloc(i)$　　ベクトル第i成分
2. 行列成分呼び出しマクロ $Mloc(n,i,j)$　　列数iの行列の(i,j)成分

[例]

- ベクトル表現
  10次元ベクトルxの宣言。
  double　x[10+1];
- ベクトルxの第3成分の呼び出し。
  x[Vloc(3)]
- 行列表現
  20×10 行列 M の宣言。
  double　M[(20+1)*(10+1)];
- 行列 M の(6,4)成分の呼び出し。
  M[Mloc(10,6,4)]

ヘッダーファイル linear.h における定数は次の通りである。

VIP　最初の成分オフセット値（ディフォルトは1）

MATSIZE　行列の行または列の最大数

MAXSIZE　行列表現用の配列の最大サイズ

MAXREPEAT　Jacobi法（回転法）のループ回数

7.2 関数リスト

関数リストを以下に載せる。

Vplus(n,Vx,Vy,Vans)

機能　　ベクトルの加算
戻り値　　なし
解説　　成分数 n のベクトル Vx とベクトル Vy を加算し、Vans に代入

Vdiff(n,Vx,Vy,Vans)

機能　　ベクトルの減算
戻り値　　なし
解説　　成分数 n のベクトル Vx とベクトル Vy を減算し、Vans に代入

Vnpro(n,k,Vx,Vans)

機能　　ベクトルのスカラー倍
戻り値　　なし
解説　　成分数 n のベクトル Vx の各成分を k 倍し、Vans に代入

Vipro(n,Vx,Vy)

機能　　ベクトルの内積
戻り値　　内積値
解説　　成分数 n のベクトル Vx とベクトル Vy の内積値を返す

Vnorm(n,Vx)

機能　　ベクトルのノルム
戻り値　　ノルム値
解説　　成分数 n のベクトル Vx のノルム値（大きさ）を返す

Vtrans(n,M,Vx,Vans)

機能　　ベクトルの線形変換
戻り値　　なし
解説　　成分数 n のベクトル Vx を n 次正方行列 M によって線形変換した結果を Vans に代入

Vcopy(n,Vx,Vans)

機能　　ベクトルのコピー
戻り値　　なし
解説　　成分数 n のベクトル Vx を Vans に複写する

Vprint(n,V)

機能　　ベクトルの表示
戻り値　　なし
解説　　成分数 n のベクトル Vx を標準出力に表示する

Mplus(n,m,Mp,Mq,Mans)

機能　　行列の加法
戻り値　　なし
解説　　(n,m)- 行列 Mp と Mq の加算結果を Mans に代入

Mdiff(n,m,Mp,Mq,Mans)

機能　　行列の減法
戻り値　　なし
解説　　(n,m)- 行列 Mp と Mq の減算結果を Mans に代入

Mpro(n,l,m,Mp,Mq,Mans)

機能　　行列の乗法
戻り値　　なし
解説　　(n,l)- 行列 Mp と (l,m)- 行列 Mq の積を (n,m)- 行列 Mans に代入

Mtr(n,Mp)

機能　　行列のトレース
戻り値　　トレース値
解説　　n 次正方行列 Mp のトレース値を Mans に代入
（トレースとは、正方行列の対角成分の和である。）

Mtp(n,m,Mp,Mans)

機能　　行列の転置
戻り値　　なし
解説　　(n,m)- 行列 Mp の転置行列を Mans に代入

Mid(n,Mans)

機能　　単位正方行列
戻り値　　なし
解説　　n 次正方行列を Mans に代入

Mdet(n,Mp)

機能　　行列式
戻り値　　行列式値
解説　　n 次正方行列 Mp の行列式の値を Mans に代入

Minv(n,Mp,Mans)

機能　　逆行列
戻り値　　行列 Mp の行列式
解説　　n 次数正方行列 Mp の逆行列を Mans に代入

Meigen(n,Mp,Vans,Mans)

機能　　行列の固有値・固有ベクトル
戻り値　　なし
解説　　n 次対称行列 Mp の固有値を n 次元ベクトル Vans に、対応する
固有ベクトルを Mans に代入。 Mans には、縦ベクトルとして代入
される。
アルゴリズムは、 Jacobi 法（回転法）を用いている。

Mcopy(n,m,Mp,Mans)

機能	行列のコピー
戻り値	なし
解説	(n,m)- 行列 Mp を Mans に複写

Mprint(n,m,Mp)

機能	行列の表示
戻り値	なし
解説	(n,m)- 行列 Mp を標準出力に表示

7.3 ソースファイル

次に線形代数ライブラリのヘッダーファイルとソースファイルを掲載する。

linear.h

```
#define VIP 1
#define Vloc(i) i + VIP -1
#define Mloc(n, i, j) (n+VIP)*(i+VIP-1)+VIP+j-1

#define MATSIZE 210
#define MAXSIZE MATSIZE * MATSIZE
#define MAXREPEAT 100

extern Vplus(), Vdiff(), Vnpro(), Vtrans(), Vprint(), Vcopy();
extern double  Vipro(), Vnorm();

extern Mplus(), Mdiff(), Mpro(), Mid(), Mtp(), Mprint(), Meigen(), Mcopy();
extern Msubmat(), loc(), jacobi(), get_sin_cos();
extern double  Mtr(), Mdet(), Minv();
```

linear.c

```
#include <stdio.h>
#include <math.h>

#include "linear.h"



/* */
/* Vector Manipulation */
/* */

Vplus(n, Vx, Vy, Vans)
  int  n;
  double  Vx[], Vy[], Vans[];
  {
    register int  i;

    for(i=1; i <= n; i++)
      Vans[Vloc(i)] = Vx[Vloc(i)] + Vy[Vloc(i)];

    return;
  }

Vdiff(n, Vx, Vy, Vans)
  int  n;
  double  Vx[], Vy[], Vans[];
  {
    register int  i;
```

```
    for(i=1; i <= n; i++)
      Vans[Vloc(i)] = Vx[Vloc(i)] - Vy[Vloc(i)];

    return;
  }

Vnpro(n, k, Vx, Vans)
  int   n;
  double  k;
  double  Vx[], Vans[];
  {
    register int  i;

    for(i = 1; i <= n; i++)
      Vans[Vloc(i)] = k * Vx[Vloc(i)];

    return;
  }

double  Vipro(n, Vx, Vy)
  int  n;
  double  Vx[], Vy[];
  {
    register int  i;
    double  value;

    value = 0.0;
    for(i = 1; i <= n; i++)
      value += Vx[Vloc(i)] * Vy[Vloc(i)];
```

```
    return value;
  }

double  Vnorm(n, Vx)
  int  n;
  double  Vx[];
  {
    return sqrt( Vipro(n, Vx, Vx) );
  }

Vtrans(n, M, Vx, Vans)
  int  n;
  double  M[], Vx[], Vans[];
  {
    register int  i, j;

    for(i = 1; i <= n; i++)
      {
Vans[Vloc(i)] = 0;
for(j = 1; j <= n; j++)
  Vans[Vloc(i)] += M[ Mloc(n, i, j) ] * Vx[Vloc(j)];
      }

    return;
  }

Vcopy(n, Vx, Vans)
  int  n;
  double  Vx[], Vans[];
  {
    register int  i;
```

```
    for(i=1; i<=n; i++)
      Vans[Vloc(i)] = Vx[Vloc(i)];

    return;
  }

Vprint(n, V)
  int  n;
  double  V[];
  {
    register int  i;

    printf("[");
    for(i=1; i < n; i++)
      printf(" %g,", V[Vloc(i)]);
    printf(" %g ]\n", V[Vloc(n)]);
    return;
  }


/* */
/* Matrix  Manipulation */
/* */

Mplus(n, m, Mp, Mq, Mans)
  int  n, m;
  double Mp[], Mq[], Mans[];
  {
    register int  i, j, k;
```

```
    for(i=1; i<=n; i++)
      for(j=1; j<=m; j++)
Mans[Mloc(m, i, j)] = Mp[Mloc(m,i,j)] + Mq[Mloc(m,i,j)];

    return;
  }

Mdiff(n, m, Mp, Mq, Mans)
  int  n, m;
  double Mp[], Mq[], Mans[];
  {
    register int  i, j;

    for(i=1; i<=n; i++)
      for(j=1; j<=m; j++)
Mans[Mloc(m, i, j)] = Mp[Mloc(m,i,j)] - Mq[Mloc(m,i,j)];

    return;
  }

Mpro(n, l, m, Mp, Mq, Mans)
  int  n, l, m;
  double  Mp[], Mq[], Mans[];
  {
    register int  i, j, k;
    double  value;

    for(i=1; i<=n; i++)
      for(j=1; j<=m; j++)
{
  value = 0.0;
```

```
  for(k=1; k<=l; k++)
    value += Mp[Mloc(l,i,k)] * Mq[Mloc(m,k,j)];
  Mans[Mloc(m,i,j)] = value;
}

    return;
  }

double  Mtr(n, Mp)
  int  n;
  double  Mp[];
  {
    register int  i;
    double  value;

    value = 0.0;
    for(i=1; i<=n; i++)
      value += Mp[Mloc(n, i, i)];

    return value;
  }

Mtp(n, m, Mp, Mans)
  int  n, m;
  double  Mp[], Mans[];
  {
    int  i, j;

    for(i=1; i<=n; i++)
      for(j=1; j<=m; j++)
Mans[Mloc(n, j, i)] = Mp[Mloc(m, i, j)];
```

```
    return;
  }

Mid(n, Mp)
  int  n;
  double  Mp[];
  {
    register int  i, j;

    for(i=1; i<=n; i++)
      for(j=1; j<=n; j++)
if( i == j )
  Mp[Mloc(n,i,j)] = 1.0;
       else
  Mp[Mloc(n,i,j)] = 0.0;

    return;
  }

Mcopy(n, m, Mp, Mans)
  int  n, m;
  double  Mp[], Mans[];
  {
    register int  i, j;

    for(i=1; i<=n; i++)
      for(j=1; j<=m; j++)
Mans[Mloc(m, i, j)] = Mp[Mloc(m, i, j)];

    return;
```

```
  }

Mprint(n, m, Mp)
  int  n, m;
  double  Mp[];
  {
    register  int  i, j;

    for(i=1; i<=n; i++)
      {
for(j=1; j<=m; j++)
  printf(" %g ", Mp[Mloc(m,i,j)]);
printf("\n\n");
      }

    return;
  }


/* Calculation of Determinant of Matrix */

double  Mdet(n, Mx)
  int  n;
  double  Mx[];
  {
    register int  i, j, k;
    double  rp, rans, *Mp;
    double  det;

    det = 1.0;
    Mp = (double *)malloc(sizeof(double) * (n+1) * (n+1));
```

```
    Mcopy(n,n, Mx, Mp);
    for(i=1; i<=n; i++)
      {
if( Mp[Mloc(n,i,i)] == 0.0 )
  {
    for(j=i+1; j<=n; j++)
      {
if( Mp[Mloc(n,j,i)] != 0.0 )
  {
    det *= -1.0;
    for(k=1; k<=n; k++)
      {
rp   = Mp[Mloc(n,i,k)];
Mp[Mloc(n,i,k)] = Mp[Mloc(n,j,k)];
Mp[Mloc(n,j,k)] = rp;
      }
    break;
  }
      }
    if( j > n )
      return 0.0;
  }

rp = Mp[Mloc(n,i,i)];
det *= rp;
for(j=1; j<=n; j++)
  {
    Mp[Mloc(n,i,j)] /= rp;
  }
for(j=1; j<=n; j++)
  {
```

```
    rp = Mp[Mloc(n,j,i)];
    if( j != i )
      for(k=1; k<=n; k++)
{
  Mp[Mloc(n,j,k)] -= rp * Mp[Mloc(n,i,k)];
}
  }
      }

    free((char *)Mp);
    return det;
  }


Msubmat(n, i, j, Mp, Mans)
  int  n, i, j;
  double  Mp[], Mans[];
  {
    register int  s, t;

    for(s=1; s<=n; s++)
      for(t=1; t<=n; t++)
Mans[Mloc(n-1, s-(s>i), t-(t>j))] = Mp[Mloc(n, s, t)];

    return;
  }


/* Calculation of Inverse Matrix */
```

```
double  Minv(n, Mx, Mans)
  int  n;
  double  Mx[], Mans[];
  {
    register int  i, j, k;
    double  rp, rans, *Mp, det;

    if( (det = Mdet(n, Mx)) == 0.0 )
      return 0;

    Mp = (double *)malloc(sizeof(double) * (n+1) * (n+1));
    Mcopy(n,n, Mx, Mp);
    Mid(n, Mans);
    for(i=1; i<=n; i++)
      {
if( Mp[Mloc(n,i,i)] == 0.0 )
  {
    for(j=i+1; j<=n; j++)
      {
if( Mp[Mloc(n,j,i)] != 0.0 )
  {
    for(k=1; k<=n; k++)
      {
rp   = Mp[Mloc(n,i,k)];
rans = Mans[Mloc(n,i,k)];
Mp[Mloc(n,i,k)] = Mp[Mloc(n,j,k)];
Mans[Mloc(n,i,k)] = Mans[Mloc(n,j,k)];
Mp[Mloc(n,j,k)] = rp;
Mans[Mloc(n,j,k)] = rans;
      }
    break;
```

```
  }
      }
  }

rp = Mp[Mloc(n,i,i)];
for(j=1; j<=n; j++)
  {
    Mp[Mloc(n,i,j)] /= rp;
    Mans[Mloc(n,i,j)] /= rp;
  }
for(j=1; j<=n; j++)
  {
    rp = Mp[Mloc(n,j,i)];
    if( j != i )
      for(k=1; k<=n; k++)
{
  Mp[Mloc(n,j,k)] -= rp * Mp[Mloc(n,i,k)];
  Mans[Mloc(n,j,k)] -= rp * Mans[Mloc(n,i,k)];
}
  }
      }

    free((char *)Mp);
    return det;
  }
```

```
/* Calculation of Eigenvalues & Eigenvectors by Jacobi's method */
```

```
Meigen(n, Mp, Vans, Mans)
  int  n;
  double  Mp[], Vans[], Mans[];
  {
    int  i, j, k;
    double  a[MAXSIZE];
    double  x[MAXSIZE];

    static double  eps = 1.0e-10;

    /* Mp -> a */
    for(i=1; i<=n; i++)
      for(j=i; j<=n; j++)
a[n*(i-1)-(i-1)*(i-2)/2+j-i] = Mp[Mloc(n,i,j)];

    /* Initialization of x */
    for(i=0; i<n; i++)
      for(j=0; j<n; j++)
if( i == j )
  x[i*n+j] = 1;
       else
  x[i*n+j] = 0;

    jacobi(a, x, n, eps, 1);

    /* a -> Vans ,  x -> Mans */
    k = 0;
    for(i = 0; i < n; i++)
      {
Vans[Vloc(i+1)] = a[k];
k += n - i;
```

```
for(j = 0; j < n; j++)
  Mans[Mloc(n,i+1,j+1)] = x[i*n+j];
      }

    return;
  }

/* Eigen Value and Eigen Vector by */
/* Jacobi method    */

loc(p, q, n)
  int  p, q, n;
  {
    int  temp;

    if( p > q )
      {
temp = p; p = q; q = temp;
      }
    return p*(2*n - p - 1)/2 + q;
  }

get_sin_cos(s, c, u, v, eps)
  double  *s, *c, u, v, eps;
  {
    double  spq;
    extern double  sqrt();

    if( u < eps )
      {
*c = 1.0 / 1.41421356237; /*   1 / sqrt(2.0)   */
```

```
*s = *c;
      }
    else
      {
spq = sqrt( u*u + v*v );
*c = sqrt( (1.0 + u/spq)/2.0 );
*s = v/(2.0 * (*c) * spq);
      }
  }

jacobi(a, vec, n, eps, getv)
  double  *a, *vec, eps;
  int  n, getv;
  {
    int  i, j, p, q, sgnv, count, flag;
    double  u, v, c, s, d, app, aqq, apq, sqrsum,
            eps1, sgn, s2, c2, sc;
    extern double  sqrt(), abs();

    if( n == 1 )
      {
vec[0] = a[0];
return 1;
      }
    eps1 = eps / (double)100.0;
    if( getv )
      for(i = 0; i < n; i++)
for(j = 0; j < n; j++)
  vec[i*n+j] = (i == j) ? 1.0: 0.0;
    /* set unit matix in vec[] */
    count = 0;
```

```
    do
      {
count += 1;
sqrsum = 0.0;
for(i = 0; i < n; i++)
  sqrsum += a[loc(i,i,n)] * a[loc(i,i,n)];
flag = 0;
for(p = 0; p < n - 1; p++)
  for(q = p+1; q < n; q++)
    /* note that p < q */
    if( abs(a[loc(p,q,n)])/sqrsum > eps )
      {
flag = 1;
app = a[loc(p,p,n)];
aqq = a[loc(q,q,n)];
apq = a[loc(p,q,n)];
d = aqq - app;
sgn = (d > 0) ? 1.0: -1.0;
u = sgn * d;
v = sgn * 2.0 * apq;
get_sin_cos(&s, &c, u, v, eps1);
s2 = s * s; c2 = c * c;
sc = 2.0 * s * c * apq;
a[loc(p,p,n)] = app*c2 + aqq*s2 - sc;
a[loc(q,q,n)] = app*s2 + aqq*c2 + sc;
a[loc(p,q,n)] = 0.0;
for(i = 0; i < n; i++)
  if( i != p && i != q )
    {
      u = a[loc(i,p,n)];
      a[loc(i,p,n)] = u*c - a[loc(i,q,n)]*s;
```

```
      a[loc(i,q,n)] = u*s + a[loc(i,q,n)]*c;
    }
if( getv )
  for(i = 0; i < n; i++)
    {
      u = vec[i*n + p];
      vec[i*n + p] = u*c - vec[i*n + q]*s;
      vec[i*n + q] = u*s + vec[i*n + q]*c;
    }
      }
      } while( flag && count < MAXREPEAT );
  }
```