情報工学実験 1 レポート
感性データの分析

AJG23055

第 8 班　牧野唯希

A. 目的

評価技法の一種である階層化意思決定法（AHP）を例に取り、C 言語 によるプログラミングを通して数値計算プログラミングの復習を行うとともに、人間の歓声に関わるデータの解析についての理解を深める。

B. 解説

実験指導書の内容と重複するため省略。レポート課題の内容についてのみ解説する。

課題は以下の通りである。

課題 1：T シャツデザインに対する分析結果報告書の作成「真夏の軽井沢に似合うのは？」

課題 2：視覚特性に関する考察

課題 3：「知見の報告」(b)一対比較表列の最大値に対応する固有ベクトルが評価基準のウェイトとして有効である理由について

C. 使用機器

OS: Windows 11 Home

Editer: Repl.it

使用言語:C 言語

D. 実験方法

＜課題 1＞

分析結果報告書の作成に向けて以下の調査を行った。

i) 感性データの観測

実験時に支持されるテーマに基づいて、パソコンのディス@プレイ上に映し出される T シャツのデザインについて一対比較を行い、一対比較表を作成した。

なお比較する要素としては、

縞模様の太さ「細い・太い」

縞模様の色「青・赤・黄」

である。

ii) プログラム作成の準備

次に一対比較行列の最大固有値と対応する固有ベクトルを算出することで、6 個の評価基準（選択肢）の重要度を示すウェイトベクトルを求める。その準備として、べき乗法を用いた固有値計算アルゴリズムに基づいて、フローチャートを作成した。

iii) C 言語によるプログラミング

作成したフローチャートをもとに行列の最大固有値及び対応する固有ベクトルを算出するプログラムを、C 言語を用いて作成した。また、作製した一対比較行列の固有ベクトルからウェイトベクトルを求めた。

これらの調査結果をもとに分析結果をまとめた報告書を作成する。

＜課題 2＞

色相による明るさの感じ方の比較を行い、一対比較表を完成させる、ただし色相は 0 から$\frac{1}{16}$刻みの 16 パターンを取り扱う。また、$\frac{3}{16},\frac{6}{16},\frac{9}{16},\frac{11}{16},\frac{15}{16}$のみを測定し、残りを Harker の方法に基づいて埋める。

その後、得られた一対比較表に対して、べき乗法を適用し、各色相のウェイトを求めることで、自身が各色相を明るいと感じる度合いを測定する。

また、RGB 値を輝度に変換する式として以下が知られている。

$$Y = 0.298912 \times R + 0.586611 \times G + 0.114478 \times B \quad (1)$$

今回の実験結果から得られる各色相の明るさのウェイトとこの式から得られる結果を比較する。

＜課題 3＞

テーマについて、文献等を用いて調べ考察する。ただしテーマは「一対比較表列の最大値に対応する固有ベクトルが評価基準のウェイトとして有効である理由について」である。

E.　実験結果

＜課題 1＞

i)　T シャツのデザインに関する一対比較表を作成した結果以下の表が得られた。

表 1T シャツのデザインに関する一対比較表

テーマ	感性データの分析					
	青，細縞	赤，細縞	黄，細縞	青，太縞	赤，太縞	黄，太縞
青，細縞		1	3	3	5	5
赤，細縞	1		3	1	1/3	1/3
黄，細縞	1/3	1/3		1/5	1/3	1/3
青，太縞	1/3	1	5		1	3
赤，太縞	1/5	3	3	1		1/3
黄，太縞	1/5	3	3	1/3	3	

ii)　つぎにべき乗法を用いた最大固有値と固有ベクトルを算出するアルゴリズムのフローチャートを以下のように作成した。

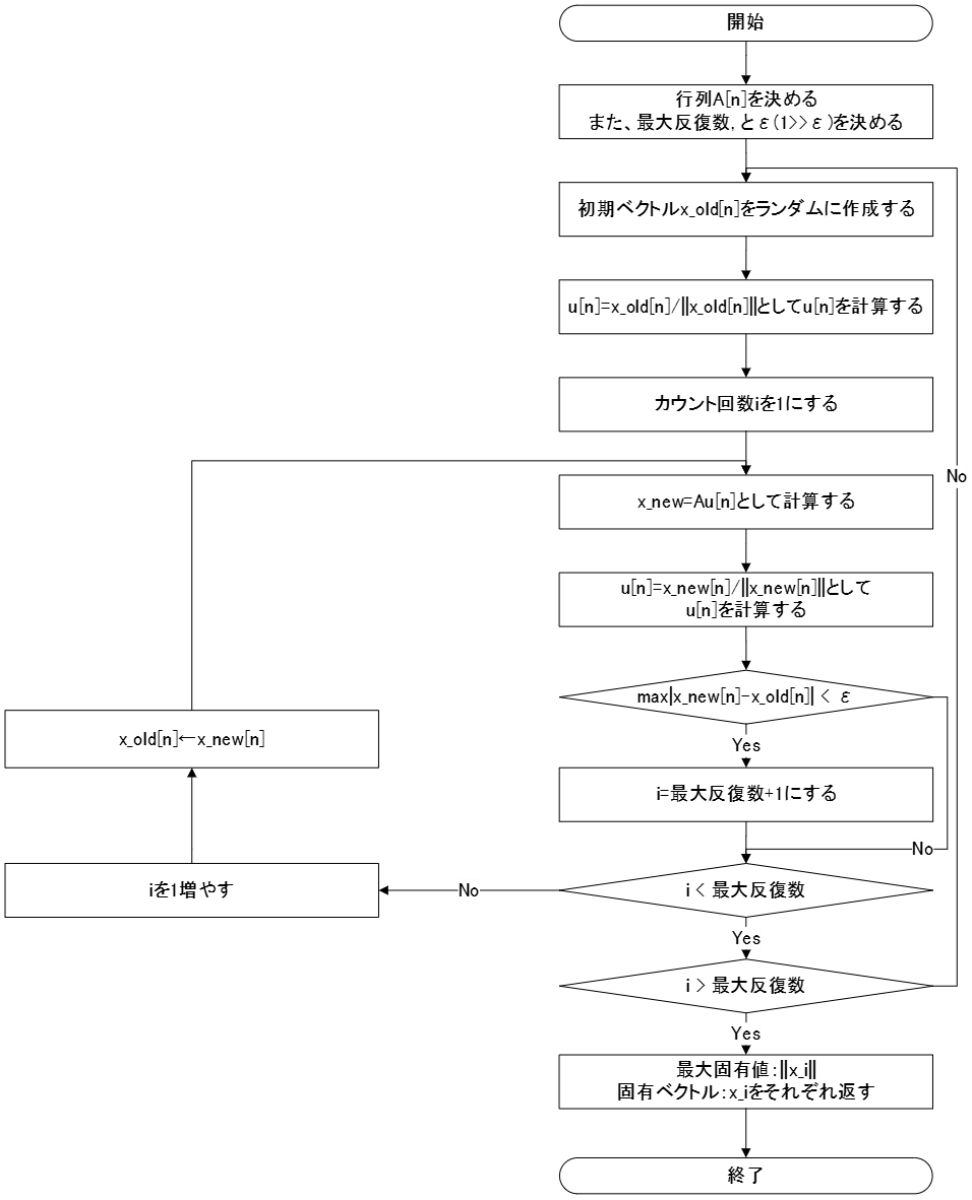

図 1 べき乗法のアルゴリズムフローチャート

iii) このフローチャートをもとに C 言語を用いたプログラムを作成した。

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define MAX_SIZE 10
#define MAX_RESTART 5  // 最大再試行回数

// ベクトルのノルムを計算する関数
```

```c
double vector_norm(double *vector, int n) {
    double norm = 0.0;
    for (int i = 0; i < n; i++) {
        norm += vector[i] * vector[i];
    }
    return sqrt(norm);
}

// ベクトルを正規化する関数
void normalize_vector(double *vector, int n) {
    double norm = vector_norm(vector, n);
    if (norm != 0.0) {
        for (int i = 0; i < n; i++) {
            vector[i] /= norm;
        }
    }
}

// 行列とベクトルの掛け算
void matrix_vector_multiply(double A[][MAX_SIZE], double *vector, double *result,
int n) {
    for (int i = 0; i < n; i++) {
        result[i] = 0.0;
        for (int j = 0; j < n; j++) {
            result[i] += A[i][j] * vector[j];
        }
    }
}

// 収束判定のための最大差分を計算
double max_difference(double *x_new, double *x_old, int n) {
    double max_diff = 0.0;
    for (int i = 0; i < n; i++) {
        double diff = fabs(x_new[i] - x_old[i]);
        if (diff > max_diff) {
            max_diff = diff;
        }
    }
    return max_diff;
}

// ランダムな初期ベクトルを生成
void generate_random_vector(double *vector, int n, int seed) {
    srand(seed);
    for (int j = 0; j < n; j++) {
        vector[j] = (double)rand() / RAND_MAX - 0.5; // -0.5 to 0.5 の範囲
    }
    normalize_vector(vector, n);
}

// べき乗法による最大固有値と固有ベクトルの計算
int power_method_with_restart(double A[][MAX_SIZE], int n, double epsilon, int
max_iterations,
                              double *eigenvalue, double *eigenvector) {

    double x_old[MAX_SIZE], x_new[MAX_SIZE], u[MAX_SIZE];
    int restart_count = 0;
    int total_iterations = 0;
```

```
    printf("べき乗法を開始します\n");
    printf("収束判定閾値: %.2e, 最大反復数: %d, 最大再試行回数: %d\n\n", epsilon,
max_iterations, MAX_RESTART);

    while (restart_count <= MAX_RESTART) {
        printf("=== 試行 %d ===\n", restart_count + 1);

        // 初期ベクトル x_old[n]をランダムに作成
        generate_random_vector(x_old, n, time(NULL) + restart_count * 1000);

        printf("初期ベクトル: ");
        for (int j = 0; j < n; j++) {
            printf("%.6f ", x_old[j]);
        }
        printf("\n");

        // u[n] = x_old[n] / ||x_old[n]||として u[n]を計算
        for (int j = 0; j < n; j++) {
            u[j] = x_old[j];
        }

        // カウント回数 i を 1 にする
        int i = 1;

        while (i < max_iterations) {
            // x_new = Au[n]として計算
            matrix_vector_multiply(A, u, x_new, n);

            // u[n] = x_new[n] / ||x_new[n]||として u[n]を計算
            for (int j = 0; j < n; j++) {
                u[j] = x_new[j];
            }
            normalize_vector(u, n);

            // max|x_new[n] - x_old[n]| < ε の判定
            double max_diff = max_difference(x_new, x_old, n);

            if (i % 10 == 0 || max_diff < epsilon) {
                printf("反復 %d: 最大差分 = %.2e\n", i, max_diff);
            }

            if (max_diff < epsilon) {
                printf("収束しました！（反復回数: %d）\n", i);

                // 固有値を返す
                double eigen = 0.0;
                for (int j = 0; j < n; j++) {
                    eigen += x_new[j] * u[j];
                }
                *eigenvalue = eigen;


                // 固有ベクトルを返す
                for (int j = 0; j < n; j++) {
```

```
                    eigenvector[j] = u[j];
                }

                total_iterations += i;
                printf("総反復回数: %d\n", total_iterations);
                return 1; // 収束成功
            }

            // x_old[n] = x_new[n]
            for (int j = 0; j < n; j++) {
                x_old[j] = x_new[j];
            }

            // i を増やす
            i++;
        }

        // i >= 最大反復数の場合
        printf("最大反復数 %d に達しました。収束しませんでした。\n", max_iterations);
        total_iterations += max_iterations;
        restart_count++;

        if (restart_count <= MAX_RESTART) {
            printf("初期値を変更して再試行します...\n\n");
        }
    }

    printf("最大再試行回数 %d に達しました。収束に失敗しました。\n", MAX_RESTART);
    printf("総反復回数: %d\n", total_iterations);
    return 0; // 収束失敗
}

int main() {
    int n = 6; // 行列のサイズ
    double A[MAX_SIZE][MAX_SIZE] = {
        {1.0, 1.0, 3.0, 3.0, 5.0, 5.0},
        {1.0, 1.0, 3.0, 1.0, 1.0/3, 1.0/3},
        {1.0/3, 1.0/3, 1.0, 1.0/5, 1.0/3, 1.0/3},
        {1.0/3, 1.0, 5.0, 1.0, 1.0, 1.0/3},
        {1.0/5, 3.0, 3.0, 1.0/3, 1.0, 1.0/3},
        {1.0/5, 3.0, 3.0, 1.0/3, 3.0, 1.0}

    };

    double epsilon = 1e-6;      // 収束判定用の閾値
    int max_iterations = 100;   // 最大反復数
    double eigenvalue;
    double eigenvector[MAX_SIZE];

    printf("行列 A:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%8.3f ", A[i][j]);
        }
        printf("\n");
```

```
    }
    printf("\n");

    int converged = power_method_with_restart(A, n, epsilon, max_iterations,
&eigenvalue, eigenvector);

    if (converged) {
        printf("\n*** 最終結果 ***\n");
        printf("最大固有値: %.6f\n", eigenvalue);
        printf("対応する固有ベクトル: ");
        for (int i = 0; i < n; i++) {
            printf("%.6f ", eigenvector[i]);
        }
        printf("\n");
    } else {
        printf("\n*** すべての試行で収束に失敗しました ***\n");
    }

    return 0;
}
```

このプログラムを実行した結果最大固有値が 6.763、対応する固有ベクトルが 0.7956, 0.2758, 0.1081, 0.2554, 0.2642, 0.3797 という結果が得られた。

```
~/.../jk3/4$ ./kadai1-3.exe
行列 A:
   1.000    1.000    3.000    3.000    5.000    5.000
   1.000    1.000    3.000    1.000    0.333    0.333
   0.333    0.333    1.000    0.200    0.333    0.333
   0.333    1.000    5.000    1.000    1.000    0.333
   0.200    3.000    3.000    0.333    1.000    0.333
   0.200    3.000    3.000    0.333    3.000    1.000

べき乗法を開始します
収束判定閾値: 1.00e-06, 最大反復数: 100, 最大再試行回数: 5

=== 試行 1 ===
初期ベクトル: 0.575589 -0.062534 0.250506 0.073094 -0.625735 0.452931
反復 10: 最大差分 = 6.95e-03
反復 20: 最大差分 = 2.03e-07
収束しました！（反復回数: 20）
総反復回数: 20

*** 最終結果 ***
最大固有値: 6.763641
対応する固有ベクトル: 0.795628 0.275847 0.108069 0.255418 0.264170 0.379712
```

これらの固有ベクトルをまとめたグラフが以下の図である。

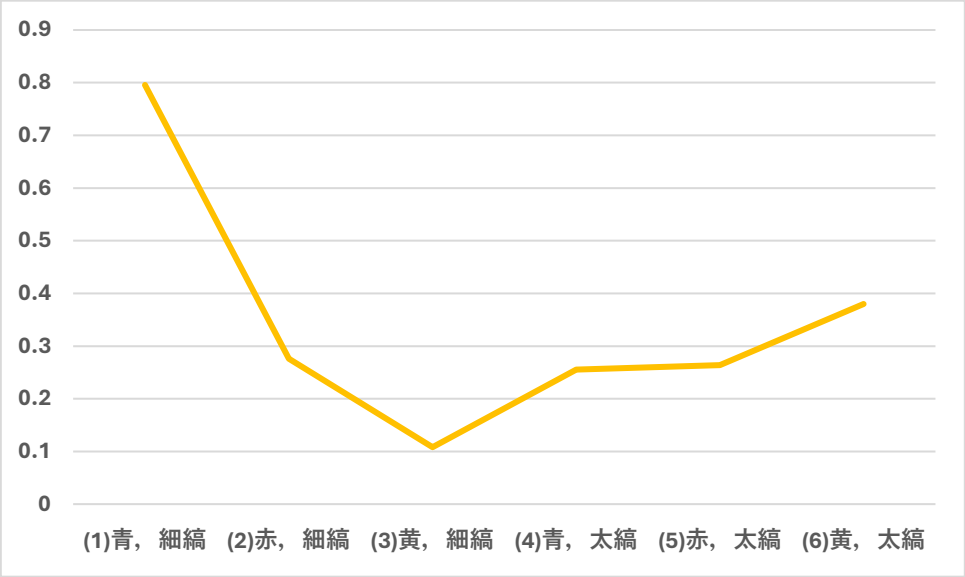

図 2T シャツデザインに関するウェイトベクトル

＜課題 2＞

特定の色相における一対比較を行い、空いている行には Harker の方法を適用させた結果、以下のような一対比較表が得られた。

Hue&Hue	$\frac{0}{16}$	$\frac{1}{16}$	$\frac{2}{16}$	$\frac{3}{16}$	$\frac{4}{16}$	$\frac{5}{16}$	$\frac{6}{16}$	$\frac{7}{16}$	$\frac{8}{16}$	$\frac{9}{16}$	$\frac{10}{16}$	$\frac{11}{16}$	$\frac{12}{16}$	$\frac{13}{16}$	$\frac{14}{16}$	$\frac{15}{16}$
$\frac{0}{16}$	11	0	0	$\frac{1}{5}$	0	0	$\frac{1}{5}$	0	0	1	0	0	2	0	0	1
$\frac{1}{16}$	0	11	0	$\frac{1}{3}$	0	0	$\frac{1}{2}$	0	0	2	0	0	5	0	0	2
$\frac{2}{16}$	0	0	11	1	0	0	1	0	0	3	0	0	6	0	0	3
$\frac{3}{16}$	5	3	1	9	0	0	2	0	0	4	0	0	8	0	0	4
$\frac{4}{16}$	0	0	0	0	12	0	1	0	0	4	0	0	7	0	0	5
$\frac{5}{16}$	0	0	0	0	0	12	1	0	0	3	0	0	5	0	0	6
$\frac{6}{16}$	5	2	1	$\frac{1}{2}$	1	1	7	0	0	4	0	0	6	0	0	6
$\frac{7}{16}$	0	0	0	0	0	0	0	13	0	2	0	0	5	0	0	5
$\frac{8}{16}$	0	0	0	0	0	0	0	0	13	3	0	0	6	0	0	8
$\frac{9}{16}$	1	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{3}$	5	0	0	3	0	0	5
$\frac{10}{16}$	0	0	0	0	0	0	0	0	0	0	14	0	1	0	0	$\frac{1}{3}$
$\frac{11}{16}$	0	0	0	0	0	0	0	0	0	0	0	14	1	0	0	$\frac{1}{5}$
$\frac{12}{16}$	$\frac{1}{2}$	$\frac{1}{5}$	$\frac{1}{6}$	$\frac{1}{8}$	$\frac{1}{7}$	$\frac{1}{5}$	$\frac{1}{6}$	$\frac{1}{5}$	$\frac{1}{6}$	$\frac{1}{3}$	1	1	3	0	0	$\frac{1}{3}$
$\frac{13}{16}$	0	0	0	0	0	0	0	0	0	0	0	0	0	15	0	$\frac{1}{2}$
$\frac{14}{16}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	2
$\frac{15}{16}$	1	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{5}$	$\frac{1}{8}$	$\frac{1}{5}$	3	5	3	2	$\frac{1}{2}$	1

この表を CSV ファイルとして保存し、べき乗法のプログラム上で読み込み、以下のプログラムで実行した。

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>

#define MAX_SIZE 16
#define MAX_RESTART 5

// 分数または整数をダブル型に変換
double parse_fraction(const char *token) {
```

```c
    if (strchr(token, '/')) {
        int num, denom;
        sscanf(token, "%d/%d", &num, &denom);
        return (double)num / denom;
    } else {
        return atof(token);
    }

}

// CSV から 16×16 行列を読み込む（1 行目・1 列目ラベルスキップ、分数対応）
int read_matrix_csv(const char *filename, double A[][MAX_SIZE], int *n) {
    FILE *fp = fopen(filename, "r");
    if (!fp) {
        printf("ファイル %s を開けませんでした。\n", filename);
        return 0;
    }
    char line[1024];
    int row = 0;
    // 1 行目スキップ（ラベル行）
    fgets(line, sizeof(line), fp);
    while (fgets(line, sizeof(line), fp) && row < MAX_SIZE) {
        int col = 0;
        char *token = strtok(line, ",\n");
        // 1 列目スキップ（ラベル列）
        token = strtok(NULL, ",\n");
        while (token && col < MAX_SIZE) {
            A[row][col] = parse_fraction(token);
            token = strtok(NULL, ",\n");
            col++;
        }
        row++;
    }
    fclose(fp);
    *n = row;
    return 1;
}

// ベクトルのノルムを計算
double vector_norm(double *vector, int n) {
    double norm = 0.0;
    for (int i = 0; i < n; i++) {
        norm += vector[i] * vector[i];
    }
    return sqrt(norm);
}

// ベクトルを正規化: u = (1/|x_old|) * x_old
void normalize_vector(double *vector, int n) {
    double norm = vector_norm(vector, n);
    if (norm != 0.0) {
        for (int i = 0; i < n; i++) {
            vector[i] /= norm;
        }
    }
}

// 行列とベクトルの掛け算: x_new = A * u
```

```c
void matrix_vector_multiply(double A[][MAX_SIZE], double *vector, double *result, int n) {
    for (int i = 0; i < n; i++) {
        result[i] = 0.0;
        for (int j = 0; j < n; j++) {
            result[i] += A[i][j] * vector[j];
        }
    }
}

// 内積計算: <x_new, u>
double inner_product(double *x_new, double *u, int n) {
    double product = 0.0;
    for (int i = 0; i < n; i++) {
        product += x_new[i] * u[i];
    }
    return product;
}

// 収束判定: max|x_new^(i) - x_old^(i)| < ε
double max_difference(double *x_new, double *x_old, int n) {
    double max_diff = 0.0;
    for (int i = 0; i < n; i++) {
        double diff = fabs(x_new[i] - x_old[i]);
        if (diff > max_diff) {
            max_diff = diff;
        }
    }
    return max_diff;
}

// ランダムベクトル生成
void generate_random_vector(double *vector, int n, int seed) {
    srand(seed);
    for (int j = 0; j < n; j++) {
        vector[j] = (double)rand() / RAND_MAX - 0.5;
    }
}

// 画像のアルゴリズムに従った固有値計算
int power_method_algorithm(double A[][MAX_SIZE], int n, double epsilon, int
max_iterations,
                           double *eigenvalue, double *eigenvector) {

    double x_old[MAX_SIZE], x_new[MAX_SIZE], u[MAX_SIZE];
    int restart_count = 0;

    printf("固有値計算を開始します（内積法）\n");
    printf("収束判定閾値: %.2e, 最大反復数: %d, 最大再試行回数: %d\n\n", epsilon,
max_iterations, MAX_RESTART);

    while (restart_count <= MAX_RESTART) {
        printf("=== 試行 %d ===\n", restart_count + 1);

        // 出発ベクトル（n 次元）を適当により、x_old とする
        generate_random_vector(x_old, n, time(NULL) + restart_count * 1000);

        printf("初期ベクトル x_old: ");
```

```
        for (int j = 0; j < n; j++) {
            printf("%.6f ", x_old[j]);
        }
        printf("\n");

        int iteration = 0;

        while (iteration < max_iterations) {
            // u = (1/|x_old|) * x_old として正規化
            for (int j = 0; j < n; j++) {
                u[j] = x_old[j];
            }
            normalize_vector(u, n);

            // x_new = A * u を計算
            matrix_vector_multiply(A, u, x_new, n);

            // 収束判定条件 max|x_new^(i) - x_old^(i)| < ε
            double max_diff = max_difference(x_new, x_old, n);

            // 10 回ごとに最大差分を表示
            if (iteration % 10 == 0 || max_diff < epsilon) {
                printf("反復 %d: 最大差分 = %.2e\n", iteration + 1, max_diff);
            }
            // 収束判定
            if (max_diff < epsilon) {
                printf("収束しました！（反復回数: %d）\n", iteration + 1);

                // 固有値 λ = <x_new, u>
                *eigenvalue = inner_product(x_new, u, n);

                // 固有ベクトル x = u
                for (int j = 0; j < n; j++) {
                    eigenvector[j] = u[j];
                }

                return 1; // 収束成功
            }

            // x_old = x_new として次の反復へ
            for (int j = 0; j < n; j++) {
                x_old[j] = x_new[j];
            }

            iteration++;
        }

        printf("最大反復数 %d に達しました。収束しませんでした。\n", max_iterations);
        restart_count++;

        if (restart_count <= MAX_RESTART) {
            printf("初期値を変更して再試行します...\n\n");
        }
    }
```

```
    printf("最大再試行回数 %d に達しました。収束に失敗しました。\n", MAX_RESTART);
    return 0; // 収束失敗
}

int main() {
    int n; // 行列のサイズ
    double A[MAX_SIZE][MAX_SIZE]; // 16×16 行列
    double epsilon = 1e-6;        // 収束判定用の閾値
    int max_iterations = 200;     // 最大反復数
    double eigenvalue;            // 最大固有値
    double eigenvector[MAX_SIZE]; // 固有ベクトル

    if (!read_matrix_csv("感性データ.csv", A, &n)) {
        printf("行列の読み込みに失敗しました。\n");
        return 1;
    }

    printf("読み込んだ行列 A（%dx%d）:\n", n, n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%8.3f ", A[i][j]);
        }
        printf("\n");
    }
    printf("\n");

    // べき乗法アルゴリズムを実行
    int converged = power_method_algorithm(A, n, epsilon, max_iterations, &eigenvalue,
eigenvector);

    if (converged) {
        printf("\n*** 最終結果 ***\n");
        printf("最大固有値 λ = <x_new, u> = %.6f\n", eigenvalue);
        printf("対応する固有ベクトル x = u: ");
        for (int i = 0; i < n; i++) {
            printf("%.6f ", eigenvector[i]);
        }
        printf("\n");
    } else {
        printf("\n*** すべての試行で収束に失敗しました ***\n");
    }

    return 0;
}
```

実行結果は以下の通りである。

```
~/.../jk3/4$ ./kadai2.exe
読み込んだ行列A（16x16）:
  11.000    0.000    0.000    0.200    0.000    0.000    0.200    0.000    0.000    1.000    0.000    0.000    2.000    0.000    0.000    1.000
   0.000   11.000    0.000    0.333    0.000    0.000    0.500    0.000    0.000    2.000    0.000    0.000    5.000    0.000    0.000    2.000
   0.000    0.000   11.000    1.000    0.000    0.000    1.000    0.000    0.000    3.000    0.000    0.000    6.000    0.000    0.000    3.000
   5.000    3.000    1.000    9.000    0.000    0.000    2.000    0.000    0.000    4.000    0.000    0.000    8.000    0.000    0.000    4.000
   0.000    0.000    0.000    0.000   12.000    0.000    1.000    0.000    0.000    4.000    0.000    0.000    7.000    0.000    0.000    5.000
   0.000    0.000    0.000    0.000    0.000   12.000    1.000    0.000    0.000    3.000    0.000    0.000    5.000    0.000    0.000    6.000
   5.000    2.000    1.000    0.500    1.000    1.000    7.000    0.000    0.000    4.000    0.000    0.000    6.000    0.000    0.000    6.000
   0.000    0.000    0.000    0.000    0.000    0.000    0.000   13.000    0.000    2.000    0.000    0.000    5.000    0.000    0.000    5.000
   0.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000   13.000    3.000    0.000    0.000    6.000    0.000    0.000    8.000
   1.000    0.500    0.333    0.250    0.250    0.333    0.250    0.500    0.333    5.000    0.000    0.000    3.000    0.000    0.000    5.000
   0.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000   14.000    0.000    1.000    0.000    0.000    0.333
   0.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000   14.000    1.000    0.000    0.000    0.200
   0.500    0.200    0.167    0.125    0.143    0.200    0.167    0.200    0.167    0.333    1.000    1.000    3.000    0.000    0.000    0.333
   0.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000   15.000    0.000    0.500
   0.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000    0.000   15.000    2.000
   1.000    0.500    0.333    0.250    0.200    0.167    0.167    0.200    0.125    0.200    3.000    5.000    3.000    2.000    0.500    1.000

固有値計算を開始します（内積法）
収束判定閾値: 1.00e-06, 最大反復数: 200, 最大再試行回数: 5

=== 試行 1 ===
初期ベクトル x_old: -0.040798 -0.226003 -0.393498 -0.135055 -0.055691 -0.129271 -0.303955 0.306186 -0.383660 0.432965 0.238930 0.490926 -0.245321 0.203744 -0.484548 -0.446877
反復   1: 最大差分 = 6.52e+00
反復  11: 最大差分 = 1.06e+00
反復  21: 最大差分 = 1.64e-01
反復  31: 最大差分 = 5.91e-02
反復  41: 最大差分 = 2.49e-02
反復  51: 最大差分 = 1.07e-02
反復  61: 最大差分 = 4.65e-03
反復  71: 最大差分 = 2.01e-03
反復  81: 最大差分 = 8.70e-04
反復  91: 最大差分 = 3.76e-04
反復 101: 最大差分 = 1.62e-04
反復 111: 最大差分 = 7.02e-05
反復 121: 最大差分 = 3.03e-05
反復 131: 最大差分 = 1.31e-05
反復 141: 最大差分 = 5.65e-06
反復 151: 最大差分 = 2.44e-06
反復 161: 最大差分 = 1.05e-06
反復 162: 最大差分 = 9.70e-07
収束しました！（反復回数: 162）

*** 最終結果 ***
最大固有値 λ = <x_new, u> = 16.312687
対応する固有ベクトル x = u: 0.087600 0.185153 0.323382 0.444981 0.373298 0.339336 0.359668 0.269415 0.394953 0.132299 0.031528 0.026973 0.046582 0.030090 0.120357 0.078996
~/.../jk3/4$
```

最大固有値が 16.31

固有ベクトルが 0.0876, 0.1852, 0.3234, 0.4450, 0.3733, 0.3394, 0.3597, 0.2694, 0.3950, 0.1323, 0.03153, 0.02697, 0.04658, 0.0301, 0.1204, 0.0790

これらの固有ベクトルをまとめたグラフが以下の図である。

図 3 色相別明るさのウェイトベクトル

次に、$\frac{1}{16}$刻みの各色相を RGB 値・YUV 値への変換を通して輝度を求めると以下の図が得られる。

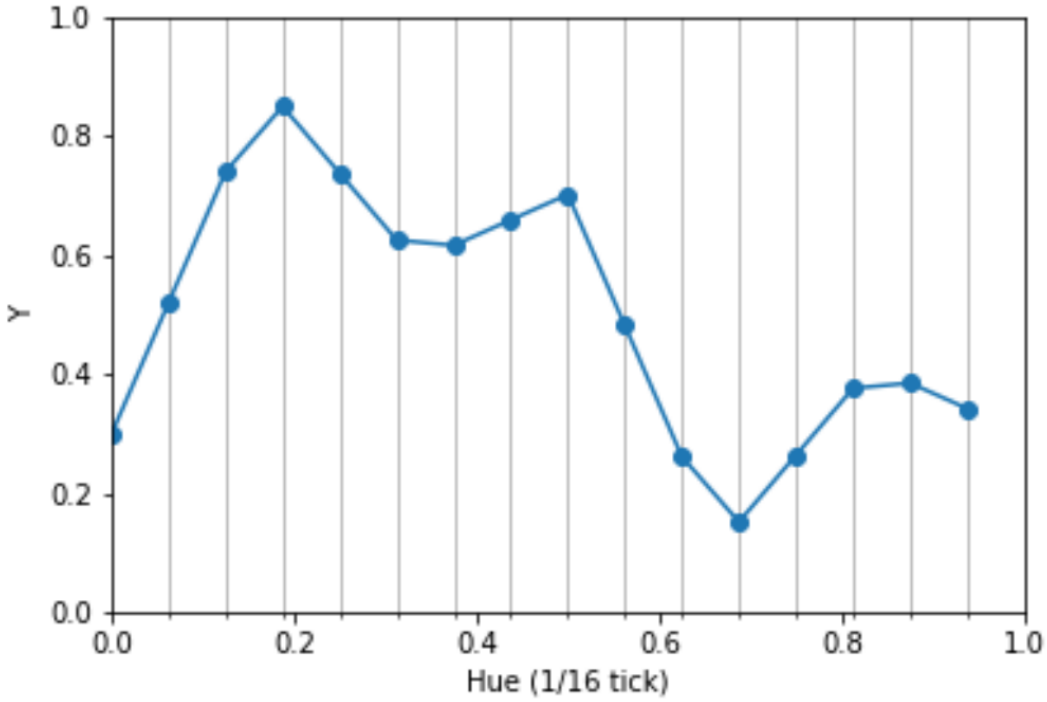

図 4RGB 値を基に作成した縦軸か輝度横軸が色相のグラフ

＜課題 3＞

課題内容で記すため省略

F. 課題内容

＜課題 1＞「T シャツのデザインに対する分析結果報告書の作成」

提案書

真夏の軽井沢に似合う T シャツデザインとしてを以下のデザインを提案

図 5 最も適した T シャツデザイン案

このたび、真夏の軽井沢に似合う T シャツデザインについて客観的な数値を用いて分析しました。その手順を最初にお伝えします。

まず、T シャツのデザインの要素を「色（赤・黄・青)」と「縞模様の太さ（太い・細い)」に分けたうえで、6 パターンのデザインを作成し、そこから 2 つのデザインを選び、どちらの方が真夏の軽井沢に似合うかを比較しました。

2 つのデザインを比較し、非常に似合う場合を 9、同じ程度に似合う場合を 1、全く似合わない場合を逆数を用いて$\frac{1}{9}$として評価し、6×6 の行列にまとめました。この時、対角成分は 1 とし、a_{ij}成分に対し、a_{ji}成分がその逆数になるように設定します。

そして、6 パターンのどれが客観的に最も似合っているかを、この行列の最大固有値と固有ベクトルというものを調べることで求めることが出来ます。

この最大固有値と固有ベクトルは、べき乗法というものを用いて C 言語でプログラムを作成し、求めました。

その結果が以下のグラフです。y 軸方向が大きいほどより似合っていることを表しており、このグラフから、青色で細い縞模様のデザインが最も夏の軽井沢に似合っていることが分かりました。

また、黄色で太い縞模様のデザインも比較的数字が大きいことから、青色で細尾い縞模様のデザインの次にこちらもに真夏の軽井沢に似合っているとわかりました。

この青色で細い縞模様が選ばれた要因としては、青色には、涼しさや冷たさを感じさせる効果があるためと考えます。一方、赤色などの暖色には、温かさを感じさせる効果があるため、夏の軽井沢というシチュエーションには適さない傾向がみられたと考えられます。

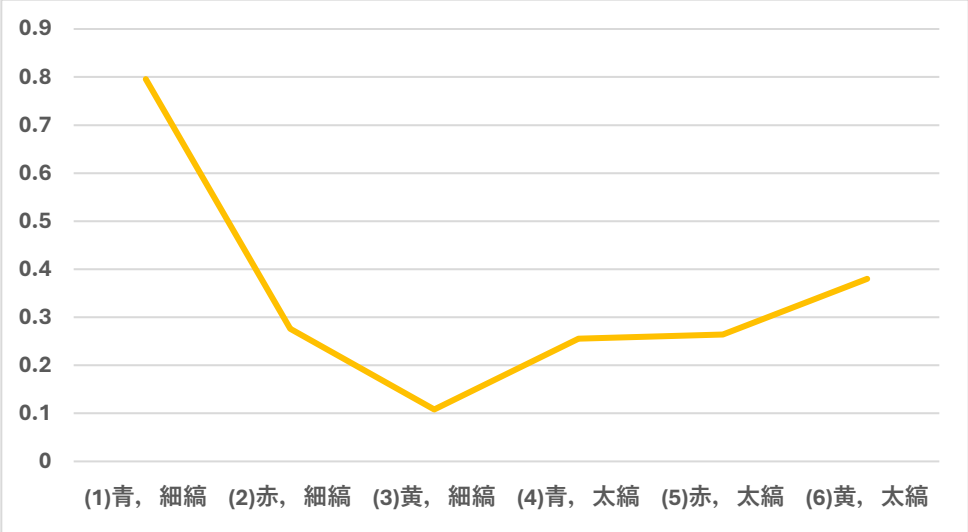

図 6T シャツデザインに関するウェイトベクトル

また、この調査の整合性について調べたところ、求められた最大固有値 6.763 において

$$CI = \frac{\lambda_{max} - n}{n - 1} = \frac{6.763 - 6}{5} = 0.1526 \quad (2)$$

という値が得られました。この値が、0.15 未満の場合は整合性があると判断できるので、整合度が高くはないものの、ある程度の整合度があることが分かりました。

これらのことから、真夏の軽井沢に似合う T シャスのデザインとして、青白で細い横島のデザインを提案します。どうかご検討のほどよろしくお願いします。

＜課題 2＞

本レポート内に含まれるため省略

＜課題 3＞

一対比較表列の最大値に対応する固有ベクトルが評価基準のウェイトとして有効である理由について

フロベニウスの定理より、非負の規約行列の主固有ベクトルは（定数倍を除いて）唯一であり、その成分は全て正とすることが出来ると言える。この性質により、AHP 行列の「最も重要な」固有値に対応する固有ベクトルは生活唯一の重みベクトルとして利用可能となる。

固有ベクトルwは

$$Aw = \lambda_{max} w \tag{3}$$

を満たす。成分ごとに見ると

$$\sum_j a_{ij} w_j = \lambda_{max} w_i \tag{4}$$

であり、各行の比較比率$a_{ij} \approx \frac{w_i}{x_j}$を$\lambda_{max}$の縮尺で最もよく近似するベクトルである。

すなわち、固有ベクトル法は一対比較行列の与えられた比率情報を総合的に反映し、判断の不一致を最小化する優れた方法であると言える。
また、整合性を検証する一貫性指標 CI を算出できる（式(2)参照）が、この式からも CI が小さいほど判断が一貫しており、固有ベクトルが表す重みも信頼性が高いことを示している。

これらのことから、一対比較行列の最大固有値に対応する固有ベクトルは、フロベニウスの定理により唯一かつ正の値を持つベクトルとして保障され、また一貫性を定量的に評価できるため、AHP における評価基準のウェイトとして有効であると考えられる。

G. 検討考察

＜課題 1＞

真夏の軽井沢で適している服装について AHP を用いて考えたが、自分の感覚を数値化させてより客観性のある情報を提示することが出来た。しかし、今回の検証では自分が比較したデータしか用いなかったため、より多くの人のデータを取得し、その平均値を用いることで、より客観的な結果を得ることが出来ると考えられる。また、今回は横縞模様に限った状態で、太さと色を変えて比較したが、それ以外の水玉模様などではどのような結果が得られるのか検討の余地があると感じた。そのように比較対象が増加していった場合に課題 2 で使用した Harker の方法をうまく活用できると感じた。

＜課題 2＞

図 3 と図 4 を比較すると、大部分に置いては同じ傾向がみられ、3/16 や 8/16 で極大値を取っている点で一致していることが分かる。一方で、12/16 の近くでは、図 4 に比べて自分の明るさのウェイトが低いことから、紫やピンクが買った色を自分はあまり明るいと感じていないと分析できる。また、縦軸の値が異なっているが、これは自分が評価をする際に明るさの違いをはっきりとしなかったために、ウェイトの差が小さくなったと考えられる。

＜課題 3＞

参考文献によると AHP を発展させた ANP という考え方があると知ったので、その方法を用いた検証を行ってみたいと感じた。ANP とは、超行列という概念を用いてその累乗の極限を用いて総合評価値を求める手法である。今回は大学の図書館で文献を探したものの、テーマに深くあてはまる文献を見つけることが出来なかったので、今後再度詳しく検証する余地があると感じた。

H. 参考文献

生方 誠希, 実験指導書：完成データの分析.pdf, 2025, p1-11

堀田敬介, 意思決定論階層分析法(AHP), 2002,
https://www.bunkyo.ac.jp/~hotta/lab/courses/2003/deciMake2003/03dmt_ahp.pdf, 2025/06/28

Hindawi, Two Proofs and One Algorithm Related to the AnalyticHierarchy Process, 2018,
https://onlinelibrary.wiley.com/doi/epdf/10.1155/2018/5241537, 2025/06/28

木下栄蔵, AHP の理論と実際, 第 4 班, 2008, p10,24