1-2 写像

例題 2 ヒストグラム

0～100点までの得点を10点幅で区切って（0～9, 10～19, …, 90～99, 100の11ランク），各ランクの度数分布（ヒストグラム）を求める．

度数を求める配列として`hist[0]`～`hist[10]`を用意する．`hist[0]`に0～9点の度数，`hist[1]`に10～19の度数，…を求めることにする．

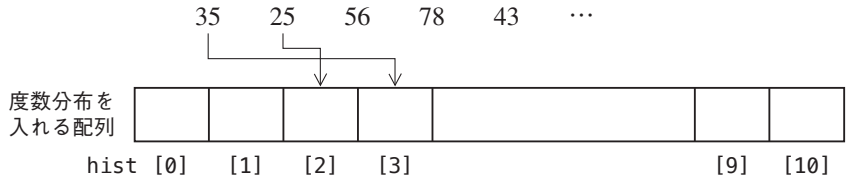


図 1.2

たとえば，35点を例にとると，これを10（度数分布の幅）で割った商の3を添字にする`hist[3]`の内容を＋1することで，度数分布のカウントアップができる．

このことは，次のように0～100点のデータ範囲を0～10の範囲に写像したと考えることができる．

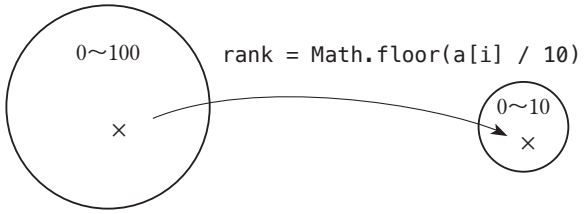


図 1.3 写像

一般に，あるデータ範囲（これを定義域という）を別のデータ範囲（これを値域という）に変換することを写像という．定義域と値域のデータ型は，異なっていてもよい．この例としては文字列から整数値への写像を行うハッシュ（**第3章3-8**）が有名．

プログラム Rei2

```
// ---------------------------------
// *       度数分布（ヒストグラム）       *
// ---------------------------------

function format(x, field, precision) {  // 書式制御
  return x.toFixed(precision).toString().padStart(field, ' ');
}

const a = [35, 25, 56, 78, 43, 66, 71, 73, 80, 90,
           0, 73, 35, 65, 100, 78, 80, 85, 35, 50];
let hist = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
for (let i = 0; i < a.length; i++) {
  let rank = Math.floor(a[i] / 10);  // 写像
  if (0 <= rank && rank <= 10) {
    hist[rank]++;
  }
}

for (let i = 0; i <= 10; i++) {
  console.log(format(i * 10, 4, 0) + ' - :  ' + hist[i]);
}
```

実行結果

```
  0 - :  1
 10 - :  0
 20 - :  1
 30 - :  3
 40 - :  1
 50 - :  2
 60 - :  2
 70 - :  5
 80 - :  3
 90 - :  1
100 - :  1
```

参考 Math.floor関数とint関数

整数化するのにp5.js専用関数のint関数を使えば簡便であるが，本書ではJavaScript標準の`Math.floor`（実数値を整数化），または`parseInt`（文字列を整数値に変換）を使用している．理由は「附録1-II　本書のプログラム書法」参照．

2-10 最小2乗法

例題 17 最小2乗法

与えられたデータ組に最も近似できる方程式$f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_mx^m$を最小2乗法により導く．

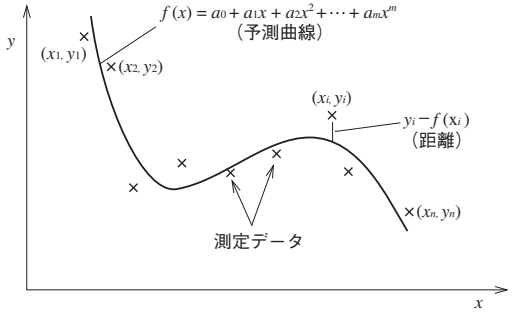


図 2.33 最小2乗法

図2.33のようなn組の測定データ(x_i, y_i) $(i = 1, 2, \cdots, n)$があるとする．このデータに沿う近似方程式として

$$f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_mx^m$$

を考え，この方程式と測定データとの距離の2乗和$\left(\sum_{i=1}^{n}(y_i - f(x_i))^2\right)$を最小となるように，係数$a_0$～$a_m$を決めるのが最小2乗法の考え方である．これは次の連立方程式を解くことにより得られる．なお，mは当てはめ曲線の次数である．

$$\begin{cases} s_0a_0 + s_1a_1 + s_2a_2 + s_3a_3 \cdots + s_ma_m = t_0 \\ s_1a_0 + s_2a_1 + s_3a_2 + s_4a_3 \cdots + s_{m+1}a_m = t_1 \\ s_2a_0 + s_3a_1 + s_4a_2 + \cdots + s_{m+2}a_m = t_2 \\ \vdots \\ s_ma_0 + s_{m+1}a_1 + \cdots + s_{2m}a_m = t_m \end{cases}$$

ただし
$$\begin{cases} s_0 = \sum_{j=1}^{n} x_j^0 \\ s_1 = \sum_{j=1}^{n} x_j^1 \\ \vdots \\ s_{2m} = \sum_{j=1}^{n} x_j^{2m} \end{cases} \quad \begin{cases} t_0 = \sum_{j=1}^{n} y_j x_j^0 \\ t_1 = \sum_{j=1}^{n} y_j x_j^1 \\ \vdots \\ t_m = \sum_{j=1}^{n} y_j x_j^m \end{cases}$$

この連立方程式はガウス・ジョルダン法で解けばよいから，次のような係数行列を作る．

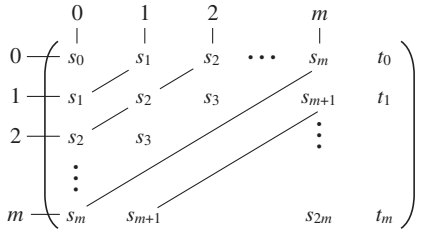


図 2.34

ここで注意してみると，係数行列の斜め方向に同じものが並んでいるので，すべての要素を計算しなくてもよいことになる．つまり，おのおのの行と列の添字の数を加えたものが等しい要素には同じ係数が入る．たとえば，s_2が入るのは2行0列，1行1列，0行2列というようにである．

プログラム Rei17_1

```
// ---------------------
// *      最小2乗法      *
// ---------------------

function format(x, field, precision) {  // 書式制御
  return x.toFixed(precision).toString().padStart(field, ' ');
}

const N = 7;  // データ数
const M = 5;  // 当てはめ次数
const x = [-3, -2, -1, 0, 1, 2, 3];
const y = [5, -2, -3, -1, 1, 4, 5];
const s = new Array(2*M + 1).fill(0);
const t = new Array(M + 1).fill(0);
const a = Array.from({ length: M + 1 }, () => new Array(M + 2));
for (let i = 0; i < N; i++) {
  for (let j = 0; j <= 2 * M; j++) {  // s0 から s2m の計算
    s[j] += Math.pow(x[i], j);
```

5-1 スタック

例題 32 プッシュ／ポップ

スタックにデータを積む関数 push とデータを取り出す関数 pop を作る.

データを棚（stack）の下部から順に積んでいき，必要に応じて上部から取り出していく方式（last in first out：後入れ先出し）のデータ構造をスタック（stack：棚）という.

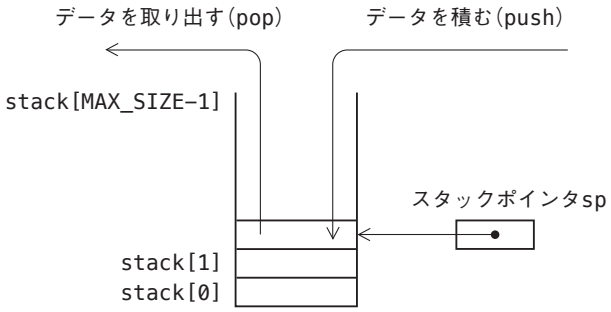


図 5.1 プッシュ／ポップ

データをスタックに積む動作を push，スタックから取り出す動作を pop と呼ぶ. スタック上のデータがどこまで入っているかをスタックポインタ sp で管理する.

データがスタックに push されるたびに sp の値は＋1され，pop されるたびに－1される.

スタックポインタ sp の初期値は最初0に設定しておき，データを push するときは現 sp が示す位置にデータを積んでから sp を＋1し，データを pop するときは，sp の値を－1してからそれが示す位置のデータを取り出すものとするとする.

したがって，sp が0の状態で pop しようとする場合は，スタックは「空の状態」であるし，sp が MAX_SIZE の状態で push しようとする場合はスタックは「溢れ状態」である.

プログラム Rei32_1

```
// --------------------
// *      スタック      *
// --------------------

let inp;
const MAX_SIZE = 1000;
const stack = new Array(MAX_SIZE);  // スタック
let sp = 0;  // スタック・ポインタ

function pushAction() {
  let data = inp.value();
  if (Push(data) === -1)
    alert('スタックは一杯');
}

function popAction() {
  let [ret, data] = Pop();
  if (ret === -1)
    alert('スタックは空');
  else
    alert(data);
}

function Push(data) {  // スタックにデータを積む
  if (sp < MAX_SIZE) {
    stack[sp] = data;
    sp++;
    return 0;
  }
  else
    return -1;  // スタックが一杯のとき
}

function Pop(data) {  // スタックからデータを取り出す
  if (sp > 0) {
    sp--;
    return [0, stack[sp]];
  }
  else
    return [-1, null];  // スタックが空のとき
}

function setup() {
  createCanvas(400, 400);
  background(220);

  inp = createInput('');
  inp.position(0, 10);
  inp.size(60);
  let bt1 = createButton('PUSH');
  bt1.position(80, 10);
```

第 8 章 グラフィックス

8-9 いろいろなリカーシブ・グラフィックス

各種リカーシブ・グラフィックスの例を以下に示す.

例題 63 C曲線

C曲線を描く.

n次のC曲線は$n-1$次のC曲線とそれぞれ90°回転させた$n-1$次のC曲線で構成される.

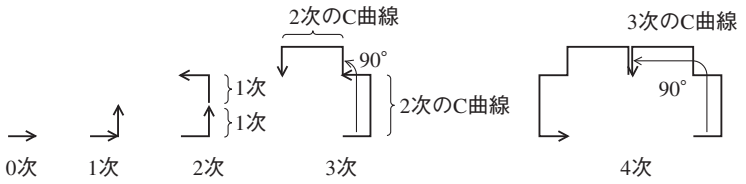
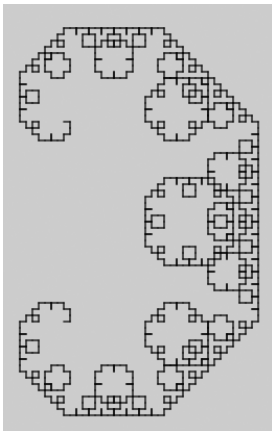


図 8.50 C曲線

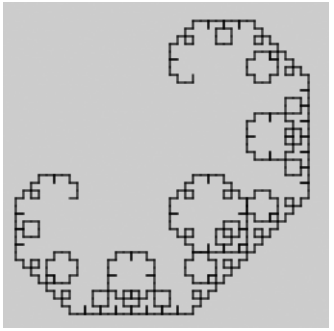
プログラム Rei63

```
// ---------------
// *    C曲線    *
// ---------------

function ccurve(n) {
  if (n === 0) {
    move(6);
  }
  else {
    ccurve(n - 1);  ← ①
    turn(90);
    ccurve(n - 1);  ← ②
    turn(-90);
  }
}

function setup() {
  createCanvas(640, 480);
  background(220);

  const n = 10;  // 次数
  ginit();
  setpoint(100, 150); setangle(0);
  ccurve(n);
}
```


実行結果



$n=10$



$n=9$

①の呼び出しを`c1()`，②の呼び出しを`c2()`と書くと3次のC曲線の再帰呼び出しは次のようになる.

9-8 板パズル

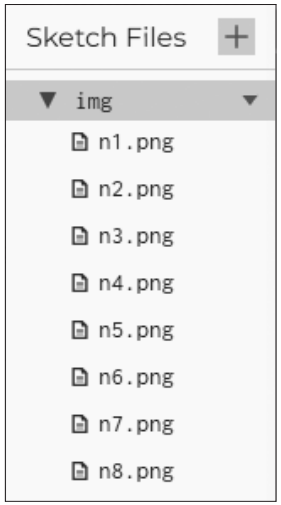
3×3のマスに1~8の板がバラバラに配置され，一か所に空きがある．板を移動できるのは，クリックした板の上下左右のどこかに空きがあるときである．板のクリックを繰り返し，1～8の順序に並べる．

板のシャッフル

配列m[][]に1~8の板を示す数字と，空きを示す「-1」を3×3のマスに初期設定する．これをシャッフルして，数字をバラバラにする．クリックした板が移動できるかチェックする際に，配列要素を超えた参照をしないように，外枠として余分な要素を作り「0」で埋める．この配列を元に，実際の板を並べる．板のイメージ（n1.png～n8.png）は60×60ピクセルとする．

図 9.26 板のイメージ

各イメージはフォルダimgを作りそこに格納しておく．



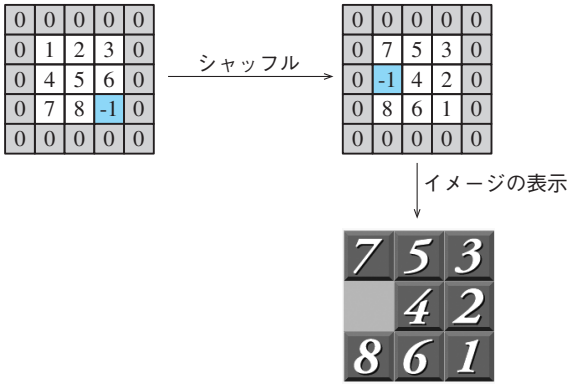


図 9.27 シャッフルとイメージの表示

例題 74 板の配置

板をシャッフルして並べる．

プログラム Rei74

```
// ------------------
// *     板パズル     *
// ------------------

const num = [null];  // 板のイメージ格納用。要素 0 は未使用
const m = [
  [0, 0, 0, 0, 0],
  [0, 1, 2, 3, 0],
  [0, 4, 5, 6, 0],
  [0, 7, 8, -1, 0],  // -1 は空きを示す
  [0, 0, 0, 0, 0]
];

function rnd(m, n) {  // m～n の乱数
  return Math.floor(Math.random()*(n - m + 1)) + m;
}

function preload() {
  for (let i = 1; i < 9; i++) {
    num.push(loadImage('img/n' + i + '.png')); // n1.png～
  }
}

function setup() {
  createCanvas(180, 180);
  for (let i = 0; i < 10; i++) {  // シャッフル
```