ななちゃんのIT教室

データ構造：行列　の巻

by nara.yasuhiro@gmail.com

ななちゃんが、行列 データ構造を
使ってみるという お話

第 0.1 版　2017 年 7 月 3 日

フリー素材
http://freeillustration.net

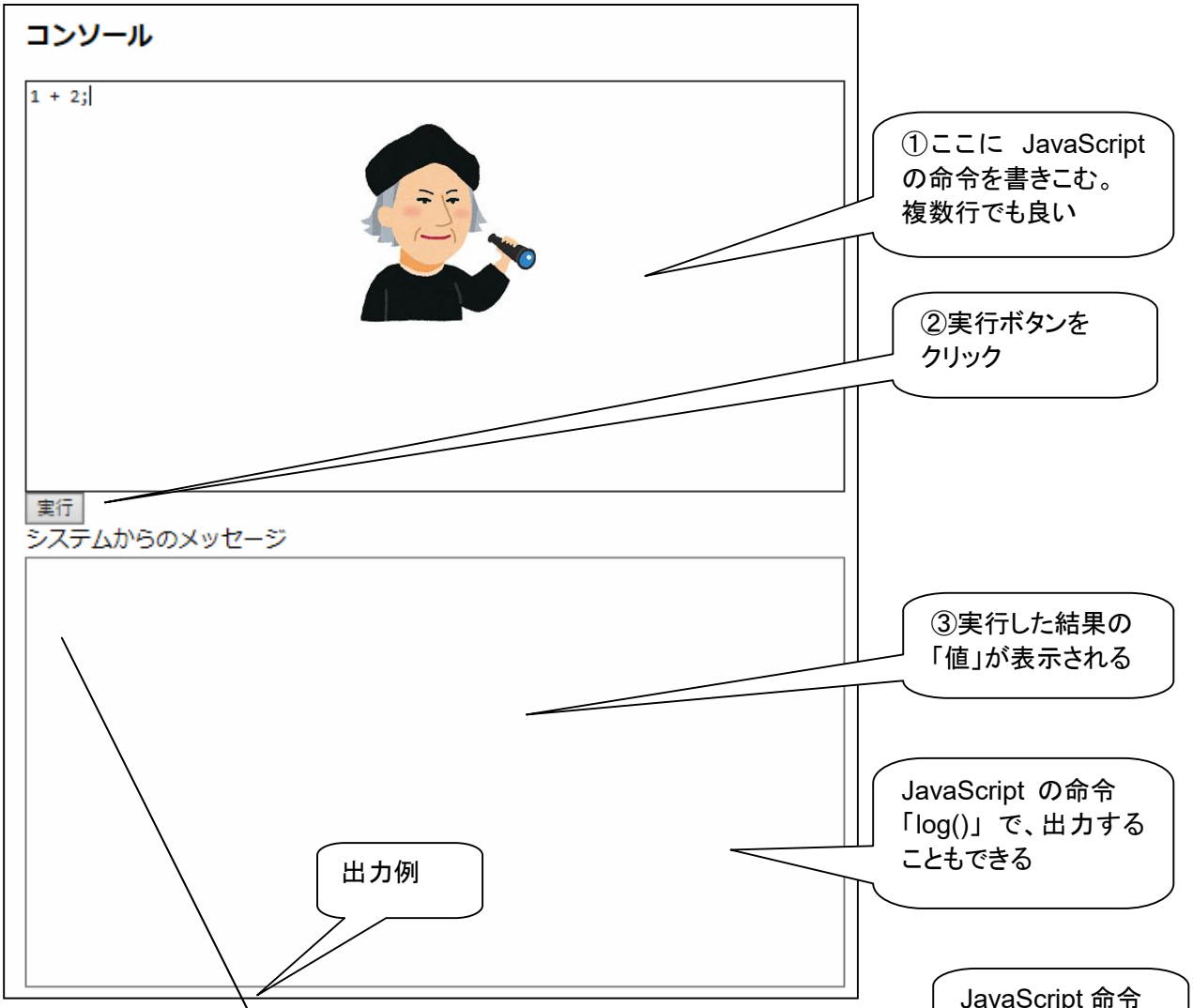
いらすとやフリー素材
http://www.irasutoya.com/

もくじ

第1回　秘密道具：マイ・コンソール

なな：クリじい、「データ構造」の勉強をするんだけど、便利な秘密道具はない？

クリ：あるぞ、あるぞ。定番秘密道具の「マイ・コンソール」。他の巻を読んでない読者のために、説明しよう。




```
<= 1 + 2;
=> Number number 3
<= "1" + "2"
=> String string "12"
<= 1;2;
=> Number number 2
<= var x = 1;
=> Undefined undefined undefined
<= x
=> Number number 1
<= var x; x= 1;
=> Number number 1
```

JavaScript 命令「1+2」を入力した

```
<= 1 + 2;
=> Number number 3
```

実行結果の「値」は 3

注意：var x = 1; の値は「undefined」

実行結果の「型」は Number

本 教材では このように 圧縮表示しています

```
1 + 2;          // Number number 3
"1" + "2"       // String string "12"
1;2;            // Number number 2
var x = 1;      // Undefined undefined undefined
x               // Number number 1
var x; x= 1;    // Number number 1
```

「型」の判定方法は 2 種類 r

「○；○」のように、複数の JavaScript 命令がある場合、一番右の命令の型、値だけ表示される

JavaScript 命令

実行結果の「型」と「値」

マイ・コンソールのプログラム。本資料のディレクトリには、行列関連メソッド定義済みのバージョンを添付しています。

```
<!DOCTYPE html>
<html>
 <head>
    <meta charset="utf-8">
    <title>コンソール</title>
 </head>
 <body>
    <h3>コンソール</h3>
    <textarea rows="19" cols="80" id=pg autofocus>1 + 2;</textarea>
    <br><input type=button onClick=go() value="実行">
    <br>システムからのメッセージ
    <br><textarea rows="20" cols="80" id=log></textarea>
    <script>
var geval = eval;

var logp = document.getElementById("log");
var pgp  = document.getElementById("pg");
var logd;

function clog(s) { logp.value += s; }
function log(s) { logd += s; }
function typeIs(obj) {
      return(Object.prototype.toString.call(obj).slice(8, -1)); }
function isPrimitive(x) {
    return (typeof x)!="object";
}
function toLiteral(x) {
    if (typeIs(x)=="Number" && isNaN(x)) return "NaN";
    if (x === Infinity)                  return "Infinity";
    if ((typeIs(x)!="Symbol")&&(-x === Infinity)) return "-Infinity";
    if (typeIs(x)=="Set") return "Set("+JSON.stringify([...x])+")";
    if (typeIs(x)=="Map") return "Map("+JSON.stringify([...x])+")";
    return JSON.stringify(x);
}
function type(x) { return "" + (typeof x); }
function isInteger(n) { return n%1 === 0; }
function keys(obj) { return Object.keys(obj); }
function go() {
 logd = "";
 try {
            var v = geval(pgp.value);
            clog("<= " + pgp.value + "\n=> "
               + typeIs(v) + " " + type(v) + " " + toLiteral(v) + "\n");
            pgp.value = "";
            logp.scrollTop = logp.scrollHeight;
            pgp.focus();
 }
 catch(e) { clog("<= " + pgp.value + "\n=>! " + e + "\n");
                pgp.value = "";
                logp.scrollTop = logp.scrollHeight;
                pgp.focus();
 }
 if (logd != "") clog(logd + "\n");
}
   </script>
 </body>
</html>
```

第2回　行列とは

なな：行列って何に使うの？

先生：まず、連立方程式。

$$\begin{aligned} w + 2x \quad\quad - z &= 1 \\ -w + x + 2y \quad &= 1 \\ 2w \quad\quad + y + z &= 1 \\ w - 2x - y + z &= 1 \end{aligned}$$

連立方程式

$$\begin{pmatrix} 1 & 2 & 0 & -1 \\ -1 & 1 & 2 & 0 \\ 2 & 0 & 1 & 1 \\ 1 & -2 & -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} w \\ x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

$$A \cdot X = R$$

行列表現

$$\begin{pmatrix} w \\ x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2 & 2 & -1 & 3 \\ -4 & -5 & 3 & -7 \\ 3 & 4 & -2 & 5 \\ -7 & -8 & 5 & -11 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

$$X = A^{-1} \cdot R$$

逆行列

$$\begin{pmatrix} w \\ x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 6 \\ -13 \\ 10 \\ -21 \end{pmatrix}$$

これが方程式の解

なな：2 変数だけのは、高校で勉強したわ。

先生：手計算をたくさんしないといけないし、計算ミスもしやすいので敬遠されがちね。でも、まさにコンピュータ向きです。細かい計算はコンピュータに任せて、結果を見ると、意外に分かりやすいものです。

先生：それから、画像回転。拡大、平行移動も可能。3D にも拡張可能。CG 分野で活用されています。

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} xr \\ yr \end{pmatrix}$$

座標回転の式

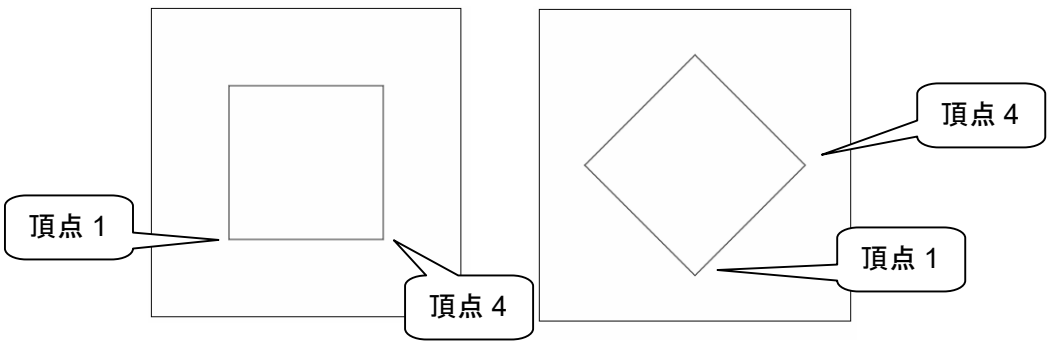
$$\begin{pmatrix} 0.71 & -0.71 \\ 0.71 & 0.71 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} xr \\ yr \end{pmatrix}$$

45°左回転の式（θ＝45°）

$$\begin{pmatrix} 0.71 & -0.71 \\ 0.71 & 0.71 \end{pmatrix} \begin{pmatrix} -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & -1 \end{pmatrix} = \begin{pmatrix} 0 & -1.42 & 0 & 1.42 \\ -1.42 & 0 & 1.42 & 0 \end{pmatrix}$$

頂点 1　頂点 4　X 座標　Y 座標

回転前　回転後



第3回　行列のコンストラクタと表示

なな：行列をどうやってプログラムにするの？

縦方向が列

横方向が行

$$A = \mathbf{A} = \underline{A} = \underline{\underline{A}} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} = (a_{ij})_{\substack{1 \le i \le m \\ 1 \le j \le n}}$$

先生：行列に含まれる行の数が m，列の数が n である時に、その行列を m 行 n 列行列、m × n 行列、mn 行列などと呼びます。行列を構成する行の数と列の数の対を型 (type) あるいはサイズといいます。m 行 n 列行列のことを (m, n)-型行列などと呼ぶこともあります。

```
function Matrix(y,x,d) {
    this.x = x;
    this.y = y;
    this.d = d.slice();
}
```

これがコンストラクタ

これが表示メソッド

```
Matrix.prototype.toString = function() {
    var s = "";
    for (var y=0; y<this.y; y++) {
        for (var x=0; x<this.x; x++) {
            s += Math.round(this.d[y*this.x+x]*100)/100 + "\t";
        }
        s += "\n";
    }
    return s;
}
```

これがマイコンソールでの使用例

太字部分が入力、それ以外は出力

```
<= var data = new Matrix(3,3,[1,2,3,
                             4,5,6,
                             7,8,9]);
=> Undefined undefined undefined
<= data;
=> Object object {"x":3,"y":3,"d":[1,2,3,4,5,6,7,8,9]}
<= log(data);
=> Undefined undefined undefined
1       2       3
4       5       6
7       8       9
```

新しいデータの定義

データをそのまま表示

toString() 表示

本資料では、このように表示します

```
var data = new Matrix(3,3,[1,2,3, 4,5,6, 7,8,9]);
data;               // Object object {"x":3,"y":3,"d":[1,2,3,4,5,6,7,8,9]}
log(data);
1       2       3
4       5       6
7       8       9
```

注意：var 文（変数定義文）、log() 文は、undefined という値を返します。

「Undefined undefined undefined]」は、型1、型2、値 の情報です。

なな：単位行列を作るには、var I = new Matrix(3,3,[1,0,0, 0,1,0, 0,0,1]) のようにするの？

先生：それでも良いけど、サイズが大きくなると大変なので、専用の関数を用意しています。

```
function IMatrix(n) {
    var I = new Array(n*n);
    for (var y=0; y<n; y++)
        for (var x=0; x<n; x++)
            I[y*n+x] = (x == y)?1:0;
    return new Matrix(n,n,I);
}
```

使用例

```
var I = IMatrix(3);
log(I);
1       0       0
0       1       0
0       0       1
```

第4回　基本的な演算子（行列和、スカラー積）

なな：行列和？。

先生：型（行数と列数）が等しいふたつの行列の「行列和」は、対応する要素の和を計算することで得られます。

$$A + B := (a_{ij} + b_{ij})_{\substack{i=1,\ldots,m,\\ j=1,\ldots,n}}$$

```
Matrix.prototype.add = function(m) {
    if ((m.x!=this.x)||(m.y!=this.y)) throw (new Error("size error"));
    var d2 = this.d.slice();
    for (var y=0; y<this.y; y++) {
        for (var x=0; x<this.x; x++) {
            d2[y*this.x+x] += m.d[y*this.x+x];
        }
    }
    return new Matrix(this.y, this.x, d2);
}
```

元のデータを壊さないように、複写

元の行列を壊さないように、新行列を生成して返す

```
var data = new Matrix(3,3,[1,2,3, 4,5,6, 7,8,9]);
log(data);
1       2       3
4       5       6
7       8       9
log(data.add(data));
2       4       6
8       10      12
14      16      18
```

使用例：自分に自分自身を足すので 2 倍になる

なな：スカラー積は？

先生：すべての要素に、スカラー値を掛けます。

$$\lambda A = (\lambda a_{ij})_{\substack{i=1,\ldots,m,\\ j=1,\ldots,n}}$$

```
Matrix.prototype.sMul = function(a) {
    var d2 = this.d.slice();
    for (var y=0; y<this.y; y++) {
        for (var x=0; x<this.x; x++) {
            d2[y*this.x+x] *= a;
        }
    }
    return new Matrix(this.y, this.x, d2);
}
```


```
var data = new Matrix(3,3,[1,2,3, 4,5,6, 7,8,9]);
log(data);
1       2       3
4       5       6
7       8       9
log(data.sMul(2));
2       4       6
8       10      12
14      16      18
```

使用例：すべての要素を二倍

第5回　少し複雑な演算子（転置、行列積）

な: 転置行列って？

先生: 行列の左上隅と右下隅を結ぶ線を軸にひっくり返えす操作です。

```
Matrix.prototype.transpose = function() {
    var d2 = this.d.slice();
    for (var y=0; y<this.y; y++) {
        for (var x=0; x<this.x; x++) {
            d2[x*this.y+y] = this.d[y*this.x+x];
        }
    }
    return new Matrix(this.x, this.y, d2);
}
```

$$ {}^tA = (a_{ji}) \underset{j=1,\ldots,n}{\scriptstyle i=1,\ldots,m,} $$

```
var data = new Matrix(3,3,[1,2,3, 4,5,6, 7,8,9]);
log(data);
1       2       3
4       5       6
7       8       9
log(data.transpose());
1       4       7
2       5       8
3       6       9
```

な: 行列積は？　行列と行列の掛け算？

先生: 前行列の行と後行列の列で、各項を掛けて足し込む操作。(m1,n1) × (m2,n2) → (m1,n2)、n1 = m2。

```
Matrix.prototype.mMul = function(m) {
    if (this.x!=m.y) {
        throw (new Error("size error"));
        return; }
    var d2 = new Matrix(this.y, m.x, new Array(m.x * this.y));
    for (var y=0; y<this.y; y++) {
        for (var x=0; x<m.x; x++) {
            var b = 0;
            for (var xx=0; xx<this.x; xx++)
                    b += this.d[y*this.x + xx] * m.d[xx*m.x + x];
            d2.d[y*m.x+x] = b;
        }
    }
    return d2;
}
```

$$ c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj} \underset{j=1,\ldots,n}{\scriptstyle i=1,\ldots,m,} $$

```
var data = new Matrix(3,3,[1,2,3, 4,5,6, 7,8,9]);
log(data);
1       2       3
4       5       6
7       8       9
var data2 = new Matrix(3,1,[1, 1, 1]);
log(data2);
1
1
1
log(data.mMul(data2));
6
15
24
```

第6回　複雑な演算子（行列式、逆行列）

なな：行列式って何？

先生：正方行列に対して計算できるスカラー値で、この値がゼロだと逆行列が計算できません。行列を写像と見た場合の、面積（体積）拡大率に相当します。プログラムでは、列交換で要素を上三角行列に集め、対角部分(左上から右下) を掛け合わせます。上三角行列の行列式は対角成分の積に等しい性質を利用します。。

```
Matrix.prototype.det = function() {
    if (this.x != this.y) { throw (new Error("size error")); return; }
    var n = this.x;
    var a = this.d.slice();
    var detv=1.0, buf, i,j,k;
    for(i=0;i<n;i++) for(j=0;j<n;j++)
        if(i < j) { buf = a[j*n+i] / a[i*n+i];
                for(k=0; k<n; k++) a[j*n+k] -= a[i*n+k] * buf; }
        for(i=0; i<n; i++) detv *= a[i*n+i];
    return new Matrix(1,1,[detv]);
}
```

なな：逆行列はどうやって計算するの？

先生：今回は、掃き出し法というアルゴリズムを使いました。。

```
Matrix.prototype.inv = function() {
    if (this.x != this.y) throw (new Error("size error"));
    var n = this.x, a = this.d.slice();
    var detv=1.0, buf, i,j,k;
    for(i=0;i<n;i++) for(j=0;j<n;j++)
        if(i < j) { buf = a[j*n+i] / a[i*n+i];
                    for(k=0; k<n; k++) a[j*n+k] -= a[i*n+k] * buf; }
        for(i=0; i<n; i++) detv *= a[i*n+i];
    if (detv == 0) throw (new Error("zero devide error"));
    var inv_a = Array(n*n);
    a = this.d.slice();
    for(i=0; i<n; i++)
        for(j=0; j<n; j++) inv_a[i*n+j] = (i==j)?1.0:0.0;
    for(i=0; i<n; i++) {
        buf = 1 / a[i*n+i];
        for(j=0; j<n; j++) {
            a[i*n+j] *= buf; inv_a[i*n+j] *= buf; }
        for(j=0; j<n; j++) {
            if(i!=j) {
                buf = a[j*n+i];
                for(k=0; k<n; k++) {
                    a[j*n+k] -= a[i*n+k] * buf;
                    inv_a[j*n+k] -= inv_a[i*n+k] * buf;
    }   }   }   }
    return new Matrix(n,n,inv_a)
}
```

行列式がゼロなら計算できないのでエラー終了

逆行列データ（inv_a）の初期値を単位行列に

元の行列（a）が単位行列になるよう、係数を掛けて行減算してゆくと、inv_a が逆行列になる

```
var datai = new Matrix(4,4,[1,2,0,-1, -1,1,2,0, 2,0,1,1, 1,-2,-1,1]);
datai.det()        // Number number -0.9999999999999991
log(datai.inv());
2       2       -1      3
-4      -5      3       -7
3       4       -2      5
-7      -8      5       -11
```

参考：http://thira.plavox.info/blog/2008/06/_c.html「行列式の値の求め方、逆行列の作り方の C 言語プログラム」

なな: 固有値って？

先生: 固有値 (eigenvalue)、固有ベクトル (eigenvector)、合わせて、固有対 (eigenpair) といいます。
「Ax = λx」を満たすゼロでないベクトル x と スカラー λが存在するとき、x を A の固有ベクトル、λ を A の固有値と呼びます。「det(λI - A) = 0」を固有方程式、特性方程式といい、これを解くことでλが求まります。解（固有値）は、一般に複素数になりますが、A が実対称行列の場合、固有方程式は永年方程式とも言われ、固有値は必ず実数となる。

なな: どうやって計算するの？

先生: A のサイズが小さい場合は、方程式を解く形になりますが、大きい（たとえば4×4以上）の場合は、繰り返し計算で誤差を小さくしてゆく方法をとるのが一般的です。実対称行列の固有値を求めるためのアルゴリズムとして、一番初歩的なのが、ヤコビ法です。

1.対角行列の固有値は対角成分そのもの

2.直交行列 P を用いて Λ = P−1 A P と書けるとき、A と Λ の固有値は一致する

という2つの定理を使います。適当な直交行列を準備して次々とAを変形してゆき、最終的に対角行列にできれば固有値が求まります。三角関数を使って直交行列を作り、A の非対角要素を次々と0に消し去ってゆきます。P が直交行列であるとは、 P の転置行列を P^T と書くとき、$PP^T = E$ が成り立つ行列のことです。
（参考:http://freak-da.hatenablog.com/entry/20120202/p1）

```
Matrix.prototype.eigen = function() {
    if (this.x != this.y) throw (new Error("size error"));
    if ((this.transpose()+"") != (this+"")) throw (new Error("not symmetric"));
    var N = this.x;
    var lmd = new Array(N), v = new Array(N*N), x;
    var a = this.d.slice();
    jacobi(N, a, lmd, v);
    var s = ""
    for (var j=0; j<N; j++){
        s += "\neigen value[" + j + "]=" + lmd[j];
        s += "\neigen vector\n";
        for(i=0; i<N; i++) {
            s += v[i+N*j] + "\n";
    }   }
    return s; }
function jacobi(N, a, lmd, v)    {
    var i, j, kmax=100, repeat, p, q;
    var eps, c, s, theta, gmax;
    var apq, app, aqq, apqmax, apj, aqj, vip, viq;
    var temp;
    gmax = 0.0;
    for (i=0; i<N; i++) {
        s = 0.0;
        for (j=i+1; j<N; j++) { s += Math.abs(a[i+N*j]); }
        if (s > gmax) gmax = s; }
    eps = 0.000001 * gmax;
    for (i=0; i<N; i++) {
        for (j=0; j<N; j++) { v[i+N*j] = 0.0; }
        v[i+N*i] = 1.0; }
    for (repeat = 1; repeat < kmax; repeat++) {
        apqmax = 0.0;
        for (p=0; p<N; p++) {
            for (q=0; q<N; q++) {
                if(p!=q) {
                    apq = Math.abs(a[p+N*q]);
                    if (apq > apqmax) apqmax = apq;
        }   }   }
        if (apqmax < eps) break;
        for (p=0; p<N-1; p++) {
            for (q=p+1; q<N; q++) {
                apq = a[p+N*q];
                app = a[p+N*p];
                aqq = a[q+N*q];
                if (Math.abs(apqmax) < eps) break;
                if (Math.abs(app-aqq) >= 1.0e-15) {
                    theta = 0.5 * Math.atan(2.0 * apq / (app-aqq));
                } else { theta = Math.PI / 4.0; }
                c = Math.cos(theta);
                s = Math.sin(theta);
                a[p+N*p] = app*c*c + 2.0*apq*c*s + aqq*s*s;
                a[q+N*q] = app*s*s - 2.0*apq*c*s + aqq*c*c;
                a[p+N*q] = 0.0;
                a[q+N*p] = 0.0;
                for (j=0; j<N; j++) {
                    if (j!=p && j!=q) {
                        apj = a[p+N*j];
                        aqj = a[q+N*j];
                        a[j+N*p] = a[p+N*j] = apj*c + aqj*s;
                        a[j+N*q] = a[q+N*j] = -apj*s + aqj*c;
                }   }
                for (i=0; i<N; i++) {
                    temp = v[i+N*p]*c+v[i+N*q]*s;
                    v[i+N*q] = -v[i+N*p] * s + v[i+N*q] * c;
                    v[i+N*p] = temp;
        }   }   }
        eps = eps * 1.05; }
    for (i=0; i<N; i++) lmd[i] = a[i+N*i];
}
```

第7回　二次元の行列演算子

なな：行列式、逆行列、固有値で、次元数が大きくなると、複雑な計算アルゴリズムが必要という話があったけど、逆に言えば、二次元とか、低次元だったら不要ということ？

先生：二次元の行列は、平面図形の写像（変形）なんかでよく使うけど、直接計算できます。多次元用の、漸化式を繰り返し使って、だんだん誤差を小さくしてゆく方法よりも制約が小さく、安定なので、重宝します。

なな：具体的には？

先生：元の行列を、下記とすると、

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

行列式は、

$$\det(A) = ad-bc$$

逆行列は、

$$\begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \div (ad-bc)$$

固有値は、

$$\det \begin{bmatrix} a-\lambda & b \\ c & d-\lambda \end{bmatrix} = 0$$

$$= (a-\lambda)(d-\lambda) - bc$$

$$= ad - a\lambda - d\lambda + \lambda^2 - bc$$

$$= \lambda^2 - (a+d)\lambda + ad - bc = 0$$

$$\lambda = ((a+d) \pm \sqrt{((a+d)^2 - 4(ad-bc))}) / 2$$

固有ベクトルは、$a-\lambda$,b

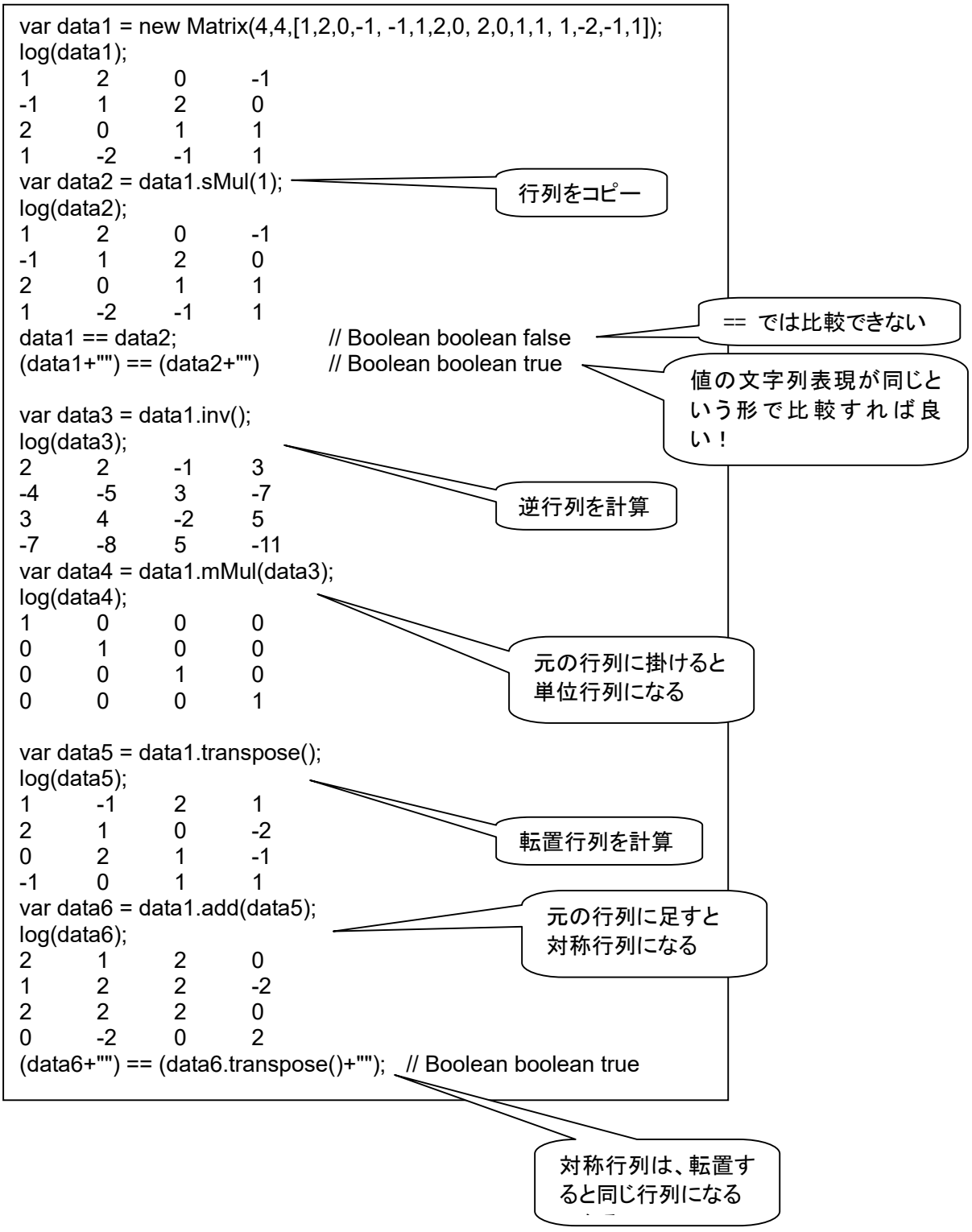
で計算できます。

```
Matrix.prototype.det2 = function() {
    if ((this.x != 2) || (this.y != 2)) throw (new Error("size error"));
    var a = this.d[0];
    var b = this.d[1];
    var c = this.d[2];
    var d = this.d[3];
    return new Matrix(1,1,[a*d-b*c]);
}

Matrix.prototype.inv2 = function() {
    if ((this.x != 2) || (this.y != 2)) throw (new Error("size error"));
    var a = this.d[0];
    var b = this.d[1];
    var c = this.d[2];
    var d = this.d[3];
    var det = a*d-b*c;
    if (det == 0) throw (new Error("zero devide"));
    return new Matrix(2,2,[d/det, -b/det, -c/det, a/det]);
}
Matrix.prototype.eigen2 = function() {
    if ((this.x != 2) || (this.y != 2)) throw (new Error("size error"));
    var lmd = [0,0], lmdi = [0,0];;
    var a = this.d[0];
    var b = this.d[1];
    var c = this.d[2];
    var d = this.d[3];
    var r = (a+d)*(a+d)-4*(a*d-b*c);
    if (r >= 0) {
        lmd[0] = (a + d + Math.sqrt(r))/2;
        lmd[1] = (a + d - Math.sqrt(r))/2;
    } else {
        lmd[0] = lmd[1] = (a + d)/2;
        lmdi[0] =     Math.sqrt(-r)/2;
        lmdi[1] = - Math.sqrt(-r)/2;
    }
    var s = ""
    for (var j=0; j<2; j++){
        s += "\neigen value[" + j + "]=" + lmd[j];
        if (r < 0) s += ((lmdi[j]>=0)?"+":"") + lmdi[j] + "i";
        s += "\neigen vector\n";
        var A = this.d[0] - lmd[j];
        var B = -this.d[1];
        var C = this.d[2];
        var D = -(this.d[3] - lmd[j]);
        s += "[" + B + "," + A; if (r < 0) s += ((lmdi[j]>=0)?"+":"") + lmdi[j] + "i";
        s += "]\n["              + D; if (r < 0) s += ((lmdi[j]>=0)?"+":"") + lmdi[j] + "i";
        s+= "," + C + "]\n"
    }
    return s;
}
```

第8回　さあ、いよいよ使ってみよう！

なな：これまでに説明でててきた行列演算は、どう使うの？

先生：行列演算のプログラムがどうなっているかより、それを使って、行列の性質をいろいろ調べたり、体験することが大切というか、目的なのよ。行列の勉強でつまづく原因の多くは、計算が面倒なことにあるの。 計算をコンピュータに任せると、行列の性質に容易に親しめるの。



```
var A = new Matrix(2,2,[1,2,3,4]);
var B = new Matrix(2,2,[5,6,7,8]);
var C = new Matrix(2,2,[9,10,11,12]);
var D = new Matrix(2,2,[13,14,15,16]);
log(A);
1       2
3       4
log(B);
5       6
7       8
log(C);
9       10
11      12
log(D);
13      14
15      16
//-------------------------------------------------------------
// 交換法則は成り立たない
log(A.mMul(B));
19      22
43      50
log(B.mMul(A));
23      34
31      46
//-------------------------------------------------------------
// 結合法則は成立
log((A.mMul(B)).mMul(C));
413     454
937     1030
log(A.mMul(B.mMul(C)));
413     454
937     1030
//-------------------------------------------------------------
// 分配則は成立
log(A.mMul(B.add(C)));
50      56
114     128
log(A.mMul(B).add(A.mMul(C)));
50      56
114     128

log(B.add(C).mMul(D));
422     452
534     572
log(B.mMul(D).add(C.mMul(D)));
422     452
534     572
var A = new Matrix(1,2,[-3,4]);
var B = new Matrix(2,1,[1,2]);
log(A);
-3      4
log(B);
1
2
log(A.mMul(B));
5
log(B.mMul(A));
-3      4
-6      8
```

A B ≒ B A

(A B) C ＝ A (B C)

A (B + C) ＝ AB + AC

(B + C) D ＝ BD + CD

(1,2)型 × (2,1)型 → (1,1)型

(2,1)型 × (1,2)型 → (2,2)型

```
var A = new Matrix(2,2,[1,0,1,2]);
log(A);
1	0
1	2
log(A.mMul(A));
1	0
3	4
// -------------------------------------------------------
var I = new Matrix(2,2,[1,0,0,1]);
log(I);
1	0
0	1
log(I.mMul(I));
1	0
0	1
// -------------------------------------------------------
var P = new Matrix(2,2,[1,0,-1,0]);
log(P);
1	0
-1	0
log(P.mMul(P));
1	0
-1	0
// -------------------------------------------------------
var Q = new Matrix(2,2,[0,0,1,1]);
log(Q);
0	0
1	1
log(Q.mMul(Q));
0	0
1	1
// -------------------------------------------------------
log(P.mMul(Q));
0	0
0	0
log(Q.mMul(P));
0	0
0	0
```

二乗すると値が変わるのが普通

二乗して変わらないのは単位行列だけか？

二乗して変わらないのは他にもある

これも二乗して変わらない

当然、ゼロ配列も二乗して変わらない

```
var r = Math.cos(Math.PI/4); r; // Number number 0.7071067811865476
var r = 1/Math.sqrt(2); r;          // Number number 0.7071067811865475
var P = new Matrix(2,2,[r,-r,r,r]); log(P);
0.71	-0.71
0.71	0.71

log(P.transpose().mMul(P));
1	0
0	1
log(P.mMul(P.transpose()));
1	0
0	1

(P.inv()+"") == (P.transpose()) // Boolean boolean true

log(P.det());
1
```

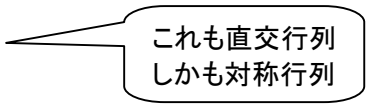
45°回転する行列

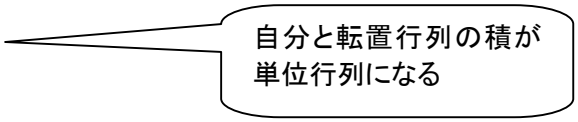
自分と転置行列の積が単位行列になるのを**直交行列**という

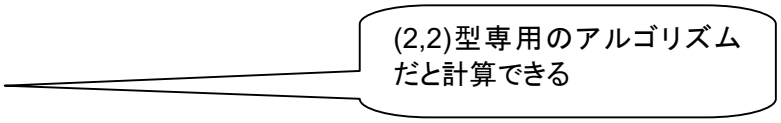
要するに、転置行列と逆行列が等しいということ

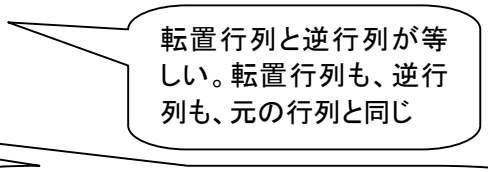
直交行列の行列式は 1 か -1 になる

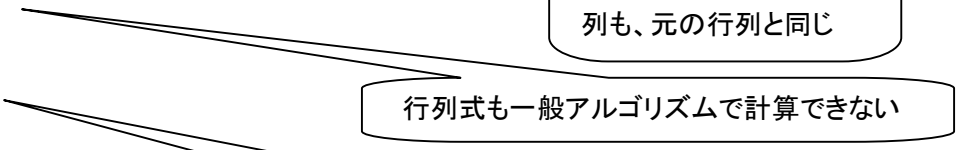
```
var Q = new Matrix(2,2,[0,1,1,0]); log(Q);
0       1
1       0

log(Q.transpose().mMul(Q))
1       0
0       1

log(Q.mMul(Q.transpose()));
1       0
0       1

log(Q.inv());
NaN     NaN
NaN     NaN

log(Q.inv2());
0       1
1       0

(Q.inv2()+"") == (Q+"") // Boolean boolean true

log(Q.det());
NaN

log(Q.det2());
-1

var R = new Matrix(2,2,[0,0.5,0.5,0]);
log(R);
0       0.5
0.5     0

var tR = R.transpose();
log(tR);
0       0.5
0.5     0

log(tR.mMul(R));
0.25    0
0       0.25

var S = new Matrix(2,2,[1,0,0,-1]); log(S);
1       0
0       -1

log(S.transpose().mMul(S));
1       0
0       1
log(S.mMul(S.transpose()));
1       0
0       1

log(S.det());
-1
```

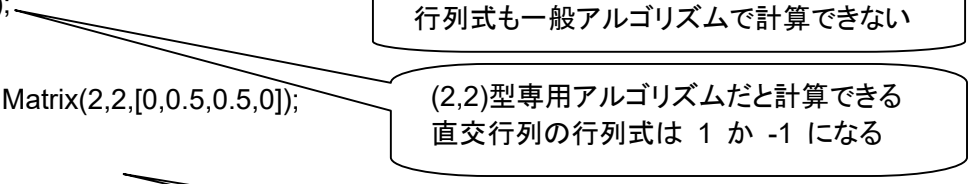


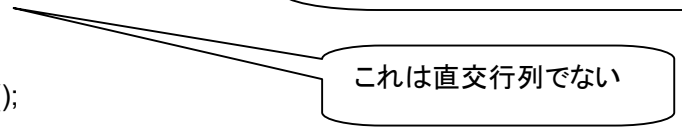


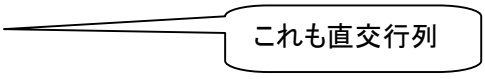












```
var A = new Matrix(2,2,[3,4,5,7]); log(A);
3       4
5       7
var IA = A.inv2(); log(IA);
7       -4
-5      3
log(A.mMul(IA));
1       0
0       1
log(IA.mMul(A));
1       0
0       1
(IA.inv2()+"") == (A+"");          // Boolean boolean true

var B = new Matrix(2,2,[2,3,3,4]); log(B);
2       3
3       4
var IB = B.inv2(); log(IB);
-4      3
3       -2

var X1 = (A.mMul(B)).inv2(); log(X1);
-43     25
31      -18
var X2 = (B.inv2()).mMul(A.inv2()); log(X2);
-43     25
31      -18
(X1+"") == (X2+"");      // Boolean boolean true
```

A の逆行列

$A \cdot A^{-1}$ は単位行列になる

$A^{-1} \cdot A$ は単位行列になる

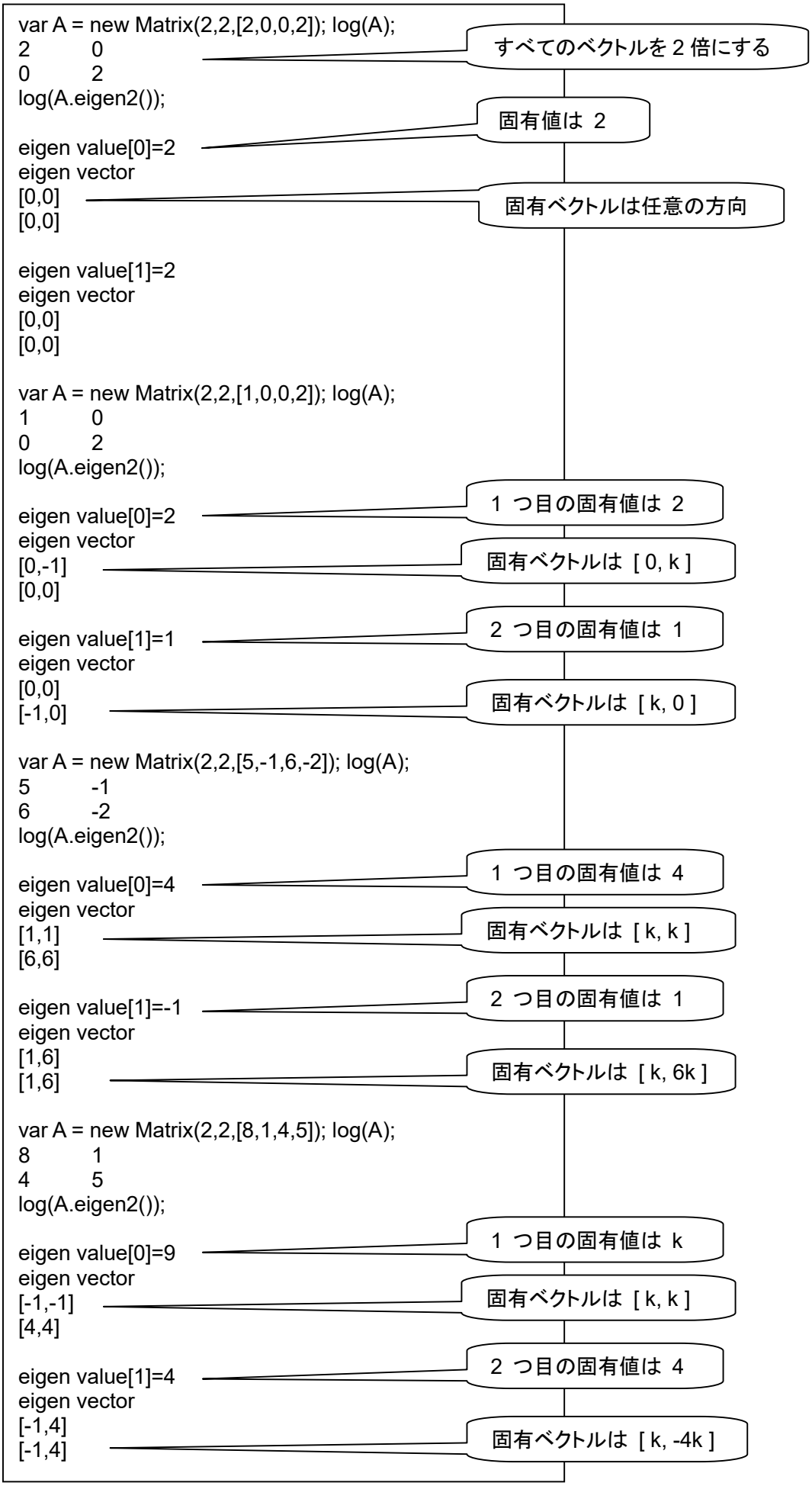
$(A^{-1})^{-1}$ は A と等しくなる

B の逆行列

$(AB)^{-1} = B^{-1} \cdot A^{-1}$

```
var A = new Matrix(2,2,[1,2,3,4]); log(A);
1       2
3       4

log(A.det());
-2

log(A.transpose().det());
-2
```

行列の行列式の値と、転置行列の行列式の値が一致することの確認。ふたつのベクトルが作る平行四辺形の面積に相当する


var A = new Matrix(2,2,[2,0,0,2]); log(A);
2 0
0 2
すべてのベクトルを 2 倍にする
log(A.eigen2());
固有値は 2
eigen value[0]=2
eigen vector
[0,0]
固有ベクトルは任意の方向
[0,0]
eigen value[1]=2
eigen vector
[0,0]
[0,0]
var A = new Matrix(2,2,[1,0,0,2]); log(A);
1 0
0 2
log(A.eigen2());
1 つ目の固有値は 2
eigen value[0]=2
eigen vector
[0,-1]
固有ベクトルは [0, k]
[0,0]
2 つ目の固有値は 1
eigen value[1]=1
eigen vector
[0,0]
[-1,0]
固有ベクトルは [k, 0]
var A = new Matrix(2,2,[5,-1,6,-2]); log(A);
5 -1
6 -2
log(A.eigen2());
1 つ目の固有値は 4
eigen value[0]=4
eigen vector
[1,1]
固有ベクトルは [k, k]
[6,6]
2 つ目の固有値は 1
eigen value[1]=-1
eigen vector
[1,6]
[1,6]
固有ベクトルは [k, 6k]
var A = new Matrix(2,2,[8,1,4,5]); log(A);
8 1
4 5
log(A.eigen2());
1 つ目の固有値は k
eigen value[0]=9
eigen vector
[-1,-1]
固有ベクトルは [k, k]
[4,4]
2 つ目の固有値は 4
eigen value[1]=4
eigen vector
[-1,4]
[-1,4]
固有ベクトルは [k, -4k]

第9回　写像としての行列

なな：行列は写像にも使えるということだったけど？

クリ：まずは、マイコンソールに、図形を描く機能を追加しよう。

```
function Canvas(size) {
    this.canvas = document.createElement('canvas');
    this.canvas.width  = (size===undefined)?400:size;
    this.canvas.height = (size===undefined)?400:size;
    this.canvas.style = "border:solid 1px";
    this.ctx = this.canvas.getContext('2d');
    document.body.appendChild(this.canvas);
}

Matrix.prototype.draw = function(canvas) {
    if (this.y != 2) throw new Error("size error");
    var N = this.x;
    var d = this.d;
    var ctx = canvas.ctx;
    ctx.beginPath();
    ctx.moveTo(cv(d[0]),canvas.canvas.width-cv(d[N]));
    for (var i=1; i<N; i++) ctx.lineTo(cv(d[i]),canvas.canvas.width-cv(d[i+N]));
    ctx.closePath();
    ctx.stroke();
    return this;
    function cv(d) { return (d + 2)*(canvas.canvas.width/4); }
}

Canvas.prototype.drawMatrix = function(matrix) {
    if (matrix.y != 2) throw new Error("size error");
    var N = matrix.x;
    var d = matrix.d;
    var ctx = this.ctx;
    var canvas = this.canvas;
    ctx.beginPath();
    ctx.moveTo(cv(d[0]),canvas.width-cv(d[N]));
    for (var i=1; i<N; i++) ctx.lineTo(cv(d[i]),canvas.width-cv(d[i+N]));
    ctx.closePath();
    ctx.stroke();
    return this;
    function cv(d) { return (d + 2)*(canvas.width/4); }
}

Canvas.prototype.clear = function() {
    var ctx = this.ctx;
    var canvas = this.canvas;
    ctx.clearRect(0,0,this.canvas.width,this.canvas.height);
    return this;
}
```

```
var c = new Canvas();
var m = new Matrix(2,4,[ -1,-1, 1 ,1,
                         -1, 1, 1,-1 ]);
m.draw(c);      とか、
c.drawMatrix(m);   で、図形が描ける。
```

m は、(2,n)型の行列で、図形を構成する各点の座標を指定する。

先生:　下記のような使い方ができます。

```
var c = new Canvas();
var f = new Matrix(2,10,[-1,1,1,-0.9,-0.9,0.3,0.3,-0.9,-0.9, -1,
                         1,1,0.9,0.9,0.05,0.05,-0.05,-0.05,-1,-1]);log(f);
-1      1       1       -0.9    -0.9    0.3     0.3     -0.9    -0.9    -1
1       1       0.9     0.9     0.05    0.05    -0.05   -0.05   -1      -1

f.draw(c);

var H = new Matrix(2,2,[0.5,0,0,0.5]);log(H);
0.5     0
0       0.5
log(H.det2());
0.25

var f2 = H.mMul(f); log(f2);
-0.5    0.5     0.5     -0.45   -0.45   0.15    0.15    -0.45   -0.45   -0.5
0.5     0.5     0.45    0.45    0.03    0.03    -0.02   -0.02   -0.5    -0.5

c.clear();
f2.draw(c);

var th = Math.PI/4; th;             // Number number 0.7853981633974483
var r = Math.cos(th); r;  // Number number 0.7071067811865476
var P = new Matrix(2,2,[r,-r,r,r]); log(P);
0.71    -0.71
0.71    0.71
log(P.det2());
1

var f3 = P.mMul(f); log(f3);
-1.41   0       0.07    -1.27   -0.67   0.18    0.25    -0.6    0.07    0
0       1.41    1.34    0       -0.6    0.25    0.18    -0.67   -1.34   -1.41

c.clear();
f3.draw(c);

var P2 = P.sMul(0.5);
var f5 = P2.mMul(f);
var b = new Matrix(2,1,[0.5,0]);log(b);
0.5
0
var v = new Matrix(1,10,[1,1,1,1,1,1,1,1,1,1]);log(v);
1       1       1       1       1       1       1       1       1       1
log(b.mMul(v));
0.5     0.5     0.5     0.5     0.5     0.5     0.5     0.5     0.5     0.5
0       0       0       0       0       0       0       0       0       0
var f6 = f5.add(b.mMul(v));
f6.draw(c);
```

「F」の頂点座標

0.5 倍する行列

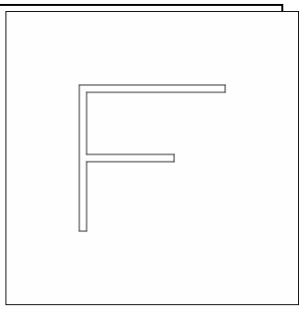
面積は 1/4

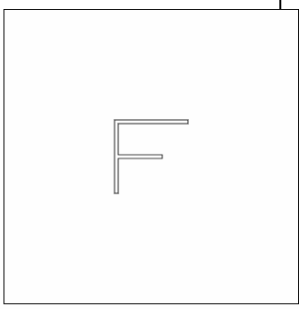
45° 左回転する行列

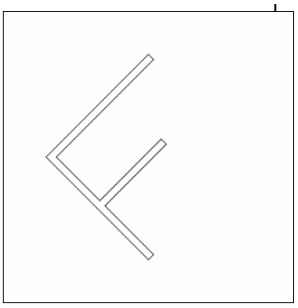
面積は 1 倍

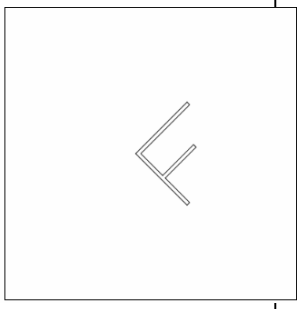
大きさを 0.5 倍

右に 0.5 移動









先生：こういうのを、「アフィン変換」といいます。

affine transformation。ラテン語 affinis（類似、関連の意）に由来。平行移動と線形変換を組み合わせた変換。線形変換は、「変換の前に直線だった場所は、変換後も直線のまま保たれる」変換。直線が変換によって曲がったりしない。「直線上に点A,B,Cが並んでいたとき、変換の前後でAB:BCの比が変化しない」。異空間なら～写像、同空間なら～変換といいます

```
new_x = s * cos(th) * x - s * sin(th) * y + tx;
new_y = s * sin(th) * x + s * cos(th) * y + ty;
```

s 拡大率
th 回転角度
tx 移動距離(X 方向)
ty 移動距離(Y 方向)

より一般的には、下記のような形になります。

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

表記法(1)

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

表記法(2)

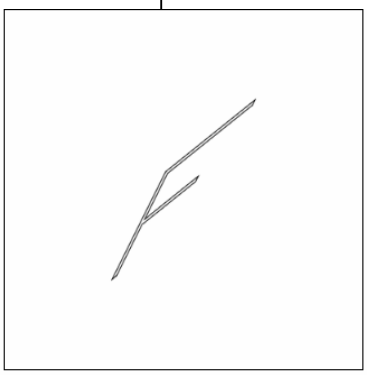
$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b & tx \\ c & d & ty \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

表記法(3)

「F」の頂点座

```
var f = new Matrix(2,10,[-1,1,   1, -0.9, -0.9,   0.3,   0.3, -0.9, -0.9, -1,
                          1, 1, 0.9, 0.9, 0.05, 0.05, -0.05, -0.05, -1,-1]);
var c = new Canvas();
var a = new Matrix(2,2,[0.5,0.3,0.4,0.6]);log(a);
0.5     0.3
0.4     0.6
a.mMul(f).draw(c);
```

写像行列



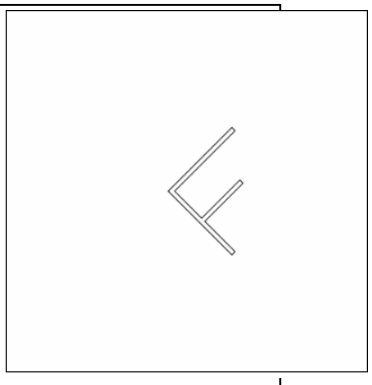
回転のパターンにとらわれず、適当な値にすると、たとえば、
こんな感じになります。拡大/縮小、回転以外に、縦横比の変化とか、
剪断とよばれる、平行四辺形のようなずれが含まれてきます。

なな：「表記法(3)」は、拡大/縮小などと　移動がひとまとめになっていて、カッコいいんだけど、どうやって使うの？

先生：こんな感じになります。

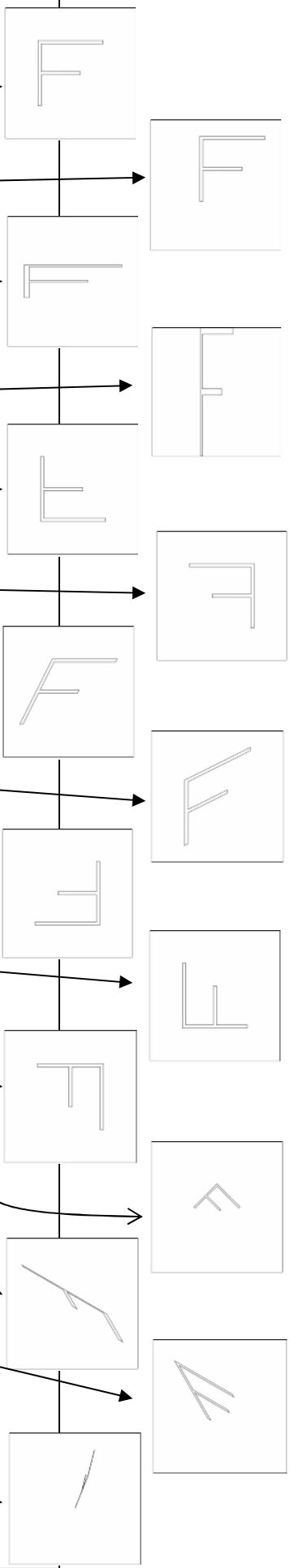
```
var th = Math.PI/4; th;
var r = 0.5*Math.cos(th); r;
var P = new Matrix(2,3,[r, -r, 0.5,
                        r,  r, 0 ]); log(P);
0.35    -0.35   0.5
0.35    0.35    0
var f = new Matrix(3,10,[-1,1,1,-0.9,-0.9,0.3,0.3,-0.9,-0.9, -1,
                         1,1,0.9,0.9,0.05,0.05,-0.05,-0.05,-1,-1,
                         1,1,1,1,1,1,1,1,1,1]);log(f);
-1      1       1       -0.9    -0.9    0.3     0.3     -0.9    -0.9    -1
1       1       0.9     0.9     0.05    0.05    -0.05   -0.05   -1      -1
1       1       1       1       1       1       1       1       1       1
var d = P.mMul(f);log(d);
-0.21   0.5     0.54    -0.14   0.16    0.59    0.62    0.2     0.54    0.5
0       0.71    0.67    0       -0.3    0.12    0.09    -0.34   -0.67   -0.71
d.draw(c);
```

写像行列



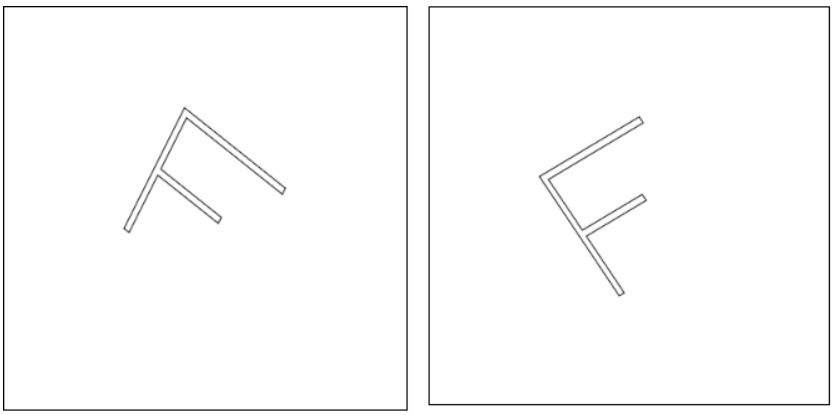
先生：写像行列にいろいろなパターンをまとめてみました。

```
function run(P) {
    var f = new Matrix(3,10,[-1,1,1, -0.9, -0.9,   0.3, 0.3, -0.9,-0.9, -1,
                              1,1,0.9,0.9,0.05,0.05,-0.05,-0.05, -1, -1,
                              1,1,   1,   1,   1,   1,      1,      1,   1,1]);
    var c = new Canvas(); var d = P.mMul(f); }
// (1) no change
var P = new Matrix(2,3,[ 1, 0, 0,
                         0, 1, 0 ]); run(P);
// (2) translate  移動
var X = 0.5, Y = 0.5;
var P = new Matrix(2,3,[ 1, 0, X,
                         0, 1, Y ]); run(P);
// (3)  縦横拡大縮小（skew、squeeze）
var W=1.5, H = 0.5;
var P = new Matrix(2,3,[ W, 0, 0,
                         0, H, 0 ]); run(P);
// (4)  縦横拡大縮小（skew、squeeze）
var k = 0.5;
var P = new Matrix(2,3,[ k,   0,   0,
                         0, 1/k, 0 ]); run(P);
// (5)  上下線対称
var P = new Matrix(2,3,[ 1,   0, 0,
                         0, -1, 0 ]); run(P);
// (6)  左右線対称
var P = new Matrix(2,3,[ -1, 0, 0,
                         0, 1, 0 ]); run(P);
// (7)  横ずれ（shear in x 横剪断）
var A = 0.5;
var P = new Matrix(2,3,[ 1, A, 0,
                         0, 1, 0 ]); run(P);
// (8)  縦ずれ（shear in y  縦剪断）
var B = 0.5;
var P = new Matrix(2,3,[ 1, 0, 0,
                        B, 1 ,0 ]); run(P);
// (9)  点対称（180°  回転）
var P = new Matrix(2,3,[ -1, 0 ,0,
                         0, -1, 0 ]); run(P);
// (10)  左 90°回転
var P = new Matrix(2,3,[ 0, -1, 0,
                         1, 0, 0 ]); run(P);
// (10B)  右 90°回転
var P = new Matrix(2,3,[ 0, 1, 0,
                        -1, 0, 0 ]); run(P);
// (11)  回転と拡大縮小（rotate）
var th = Math.PI/4, r = 0.5;
var P = new Matrix(2,3,[ r*Math.cos(th),   r*Math.sin(th),   0,
                        -r*Math.sin(th),   r*Math.cos(th),   0 ]); run(P);
// (12)  縦横独立回転（例：縦は左回転、横は右回転）
var r = 0.5, s = 1.5;
var th = Math.PI/3, ph = Math.PI*2/3;
var P = new Matrix(2,3,[ r*Math.cos(th),   s*Math.sin(ph),   0,
                        -r*Math.sin(th),   s*Math.cos(ph),   0]); run(P);
// (12B)  縦横独立回転（例：縦線は左回転、横線は右回転）
var r = 1, s = 1;
var th = Math.PI/6, ph = -Math.PI/6;
var P = new Matrix(2,3,[ r*Math.cos(th), s*Math.sin(ph), 0,
                        -r*Math.sin(th), s*Math.cos(ph), 0]); run(P);
// (20)  完全独立、自由
var r = 0.5, s = 1.5;
var th = Math.PI/3, ph = Math.PI*2/3;
var P = new Matrix(2,3,[ 0.1, 0.2, 0.3,
                         0.4, 0.5, 0.6 ]); run(P);
```



先生：写像行列を転置してみましょう。縮小効果は同じで、回転角度が反転します。

```
var f = new Matrix(2,10,[-1,1,    1, -0.9, -0.9,    0.3,    0.3, -0.9, -0.9, -1,
                                1, 1, 0.9, 0.9, 0.05, 0.05, -0.05, -0.05, -1,-1]);
var c = new Canvas();
var a = new Matrix(2,2,[0.5,0.3,-0.4,0.6]);log(a);
0.5       0.3
-0.4      0.6
a.mMul(f).draw(c);
var c = new Canvas();
var a = new Matrix(2,2,[0.5,0.3,-0.4,0.6]);
a.transpose().mMul(f).draw(c);
```



先生：こんどは、写像行列の逆行列を作ってみましょう。右回転→左回転、縮小→拡大に変わります。

```
var f = new Matrix(2,10,[-1,1,    1, -0.9, -0.9,    0.3,    0.3, -0.9, -0.9, -1,
                                1, 1, 0.9, 0.9, 0.05, 0.05, -0.05, -0.05, -1,-1]);
var c = new Canvas();
var a = new Matrix(2,2,[0.5,0.3,-0.4,0.6]);
a.inv2().mMul(f).draw(c);
```