

glmMatrix

2015 年度 情報可視化論

陰山 聡

2015.06.16

glmMatrix の紹介

行列ライブラリ

- WebGL では JavaScript で行列を扱う必要がある
- モデル変換、ビュー変換、射影変換
- 4x4 または 3x3 の行列演算
- JavaScript で行列を扱うライブラリは複数ある
- その中で、ここでは glMatrix を使う

glMatrix.js

<http://glmatrix.net>

最新バージョンは 2.0

この講義ではバージョン 0.9.6 を使用

以下の URL からソースコードを入手

<http://www.research.kobe-u.ac.jp/csi-viz/members/kageyama/lec>

自分の WebGL ソースコードと同じディレクトリに glMatrix.js という名前で保存

glMatrix.js Copyright

```
/*
 * glMatrix.js – High performance matrix and vector
 *   operations for WebGL
 * version 0.9.6
 */

/*
 * Copyright (c) 2011 Brandon Jones
 *
 * This software is provided 'as-is', without any express or
 *   implied
 * warranty. In no event will the authors be held liable for
 *   any damages
 * arising from the use of this software.
 *
 * Permission is granted to anyone to use this software for
 *   any purpose ,
 * including commercial applications , and to alter it and
 *   redistribute it
 * freely , subject to the following restrictions:
```

glmMatrix のベクトルと行列

vec3, vec4, mat3, mat4, quat4

全て 型付き配列 Float32Array

glmMatrix の関数

vec3.{create/set/add/subtract/negate/scale/normalize/cross/length/
dot/direction/lerp/str}

mat3.{create/set/identity/transpose/toMat4/str}

mat4.{create/set/identity/transpose/determinant/inverse/
toRotationMat/toMat3/toInverseMat3/multiply/multiplyVec3/
multiplyVec4/translate/scale/rotate/rotateX/rotateY/rotateZ/
frustum/perspective/ortho/lookAt/str}

quat4.{create/set/calculateW/inverse/length/normalize/multiply/
multiplyVec3/toMat3/toMat4/slerp/str}

例 : glMatrix_exercise_00.html

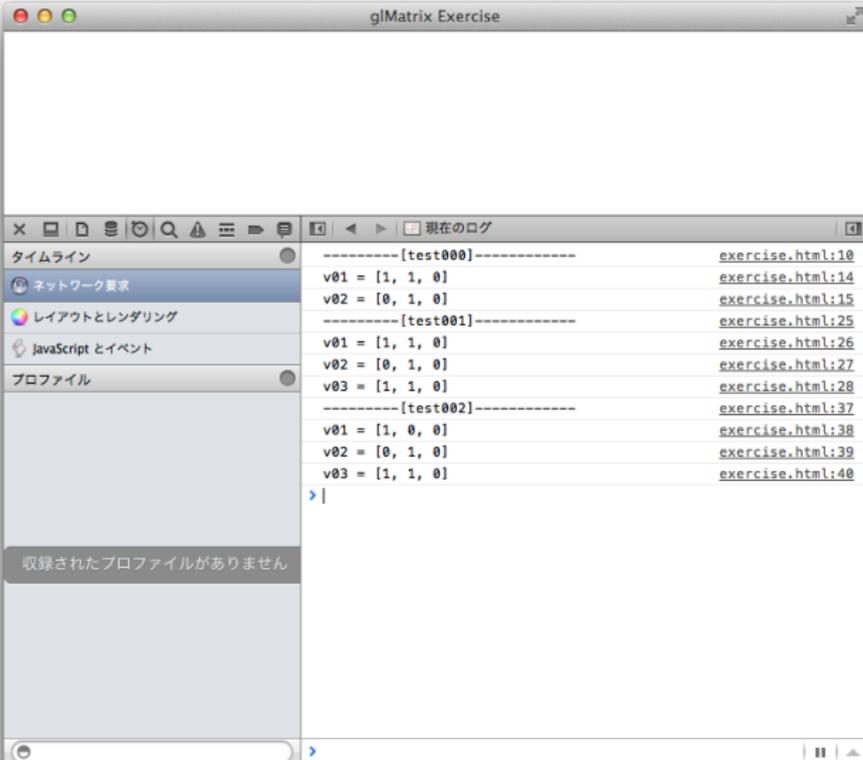
```
<!DOCTYPE HTML>
<html lang="en">
<head>
<title>glMatrix Exercise</title>
<meta charset="utf-8">
<script type="text/javascript" src="glMatrix.js"></script>
<script type="text/javascript">

function test000() {
  console.log("-----[test000]-----");
  var v01 = vec3.create([1.0, 0.0, 0.0]);
  var v02 = vec3.create([0.0, 1.0, 0.0]);
  vec3.add(v01, v02);
  console.log(" v01 = " + vec3.str(v01));
  console.log(" v02 = " + vec3.str(v02));
}
```

```
test001();  
}  
  
function test001() {  
  var v01 = vec3.create([1.0, 0.0, 0.0]);  
  var v02 = vec3.create([0.0, 1.0, 0.0]);  
  var v03 = vec3.create();  
  
  v03 = vec3.add(v01, v02);  
  console.log("-----[test001]-----");  
  console.log(" v01 = " + vec3.str(v01));  
  console.log(" v02 = " + vec3.str(v02));  
  console.log(" v03 = " + vec3.str(v03));  
  test002();  
}
```

```
function test002 () {  
  var v01 = vec3.create([1.0, 0.0, 0.0]);  
  var v02 = vec3.create([0.0, 1.0, 0.0]);  
  var v03 = vec3.create();  
  vec3.add(v01, v02, v03);  
  console.log("-----[test002]-----");  
  console.log(" v01 = " + vec3.str(v01));  
  console.log(" v02 = " + vec3.str(v02));  
  console.log(" v03 = " + vec3.str(v03));  
}  
  
</script>  
</head>
```

glMatrix_exercise_00.html : 結果



The screenshot shows a web browser window titled "glMatrix Exercise". The console output is as follows:

```
-----[test000]-----  
v01 = [1, 1, 0] exercise.html:10  
v02 = [0, 1, 0] exercise.html:14  
-----[test001]-----  
v01 = [1, 1, 0] exercise.html:25  
v02 = [0, 1, 0] exercise.html:26  
v03 = [1, 1, 0] exercise.html:27  
-----[test002]-----  
v01 = [1, 0, 0] exercise.html:37  
v02 = [0, 1, 0] exercise.html:38  
v03 = [1, 1, 0] exercise.html:39  
> |
```

glmMatrix における引数のルール

ベクトルの足し算

第一引数が更新される

```
vec3.add(v01, v02);    // v01 + v02 ⇒ v01
```

左辺と第一引数が更新される

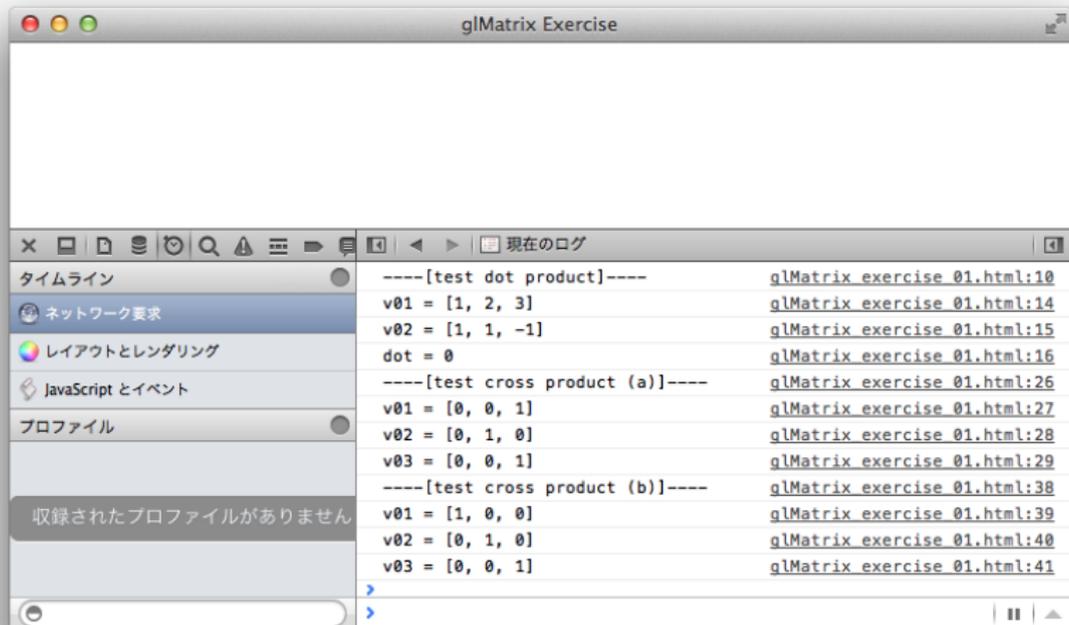
```
v03 = vec3.add(v01, v02);    // v01 + v02 ⇒ v01 & v03
```

第一引数を更新したくない場合はこうする：

```
vec3.add(v01, v02, v03);    // v01 + v02 ⇒ v03
```

ベクトルの内積と外積

glMatrix_exercise_01.html



```
glMatrix Exercise  
----[test dot product]----  
v01 = [1, 2, 3]  
v02 = [1, 1, -1]  
dot = 0  
----[test cross product (a)]----  
v01 = [0, 0, 1]  
v02 = [0, 1, 0]  
v03 = [0, 0, 1]  
----[test cross product (b)]----  
v01 = [1, 0, 0]  
v02 = [0, 1, 0]  
v03 = [0, 0, 1]
```

The screenshot shows a browser window titled "glMatrix Exercise" with a console panel at the bottom. The console displays the following output:

- [test dot product]----
 - v01 = [1, 2, 3]
 - v02 = [1, 1, -1]
 - dot = 0
- [test cross product (a)]----
 - v01 = [0, 0, 1]
 - v02 = [0, 1, 0]
 - v03 = [0, 0, 1]
- [test cross product (b)]----
 - v01 = [1, 0, 0]
 - v02 = [0, 1, 0]
 - v03 = [0, 0, 1]

ベクトルの内積

glmMatrix_exercise_01.html

```
var v01 = vec3.create([1.0, 2.0, 3.0]);  
var v02 = vec3.create([1.0, 1.0, -1.0]);  
var ans = vec3.dot(v01, v02);      // ans = 0
```

ベクトルの外積 (a)

glMatrix_exercise_01.html

```
var v01 = vec3.create([1.0, 0.0, 0.0]);  
var v02 = vec3.create([0.0, 1.0, 0.0]);  
var v03 = vec3.create();  
v03 = vec3.cross(v01, v02);    // v01 は更新される (=v03)
```

ベクトルの外積 (b)

glMatrix_exercise_01.html

```
var v01 = vec3.create([1.0, 0.0, 0.0]);  
var v02 = vec3.create([0.0, 1.0, 0.0]);  
var v03 = vec3.create();  
vec3.cross(v01, v02, v03);      // v01 は更新されない。
```

演習

演習

- glMatrix_exercise_00.html と
- glMatrix_exercise_01.html を動かしてみよう。

ブラウザでコンソールを開くこと。

ベクトルの規格化 (a)

glMatrix_exercise_02.html

長さを 1 にする

```
var v01 = vec3.create([3.0, 4.0, 0.0]);  
vec3.normalize(v01);
```

ベクトルの規格化 (b)

glMatrix_exercise_02.html

```
var v01 = vec3.create([3.0, 4.0, 0.0]);  
var v02 = vec3.create();  
vec3.normalize(v01, v02);  
    // v01 を規格化したものが v02 に入る。  
    // v01 は更新されない。
```

ベクトルの大きさ

glmMatrix_exercise_03.html

```
var v01 = vec3.create([3.0, 4.0, 12.0]);  
var amp = vec3.length(v01);
```

ベクトルのコピー

glMatrix_exercise_03.html

```
var v01 = vec3.create([1.0, 2.0, 3.0]);  
var v02 = vec3.create();  
vec3.set(v01,v02)  
    // v01 の 3 成分全てが v02 にコピーされる
```

ベクトルの符号反転 (a)

glmMatrix_exercise_03.html

```
vec3.negate(v01);  
// v01 の符号を変える
```

ベクトルの符号反転 (b)

glMatrix_exercise_03.html

```
var v01 = vec3.create([3.0, 4.0, .0]);  
var v02 = vec3.create();  
vec3.negate(v01, v02);  
    // v01 の符号を変えたものが v02 に入る。  
    // v01 は変わらない。
```

ベクトルの引き算

glMatrix_exercise_03.html

add と同じ

`vec3.add` \Rightarrow `vec3.subtract`

とすればいい。

add と同様に 3 通りの書き方がある。

ベクトルの拡大縮小 (a)

glMatrix_exercise_03.html

```
var v01 = vec3.create([1.0, 2.0, 3.0]);  
var s = 0.1;  
vec3.scale(v01, s);  
    // v01 が s 倍される。
```

ベクトルの拡大縮小 (b)

glMatrix_exercise_03.html

```
var v01 = vec3.create([1.0, 2.0, 3.0]);
var v02 = vec3.create();
var s = 0.1;
vec3.scale(v01, s, v02);
// v01 を s 倍したものが v02 に入る。
// v01 は変わらず。
```

二つの点を結ぶ単位方向ベクトル (a)

glMatrix_exercise_03.html

```
var v01 = vec3.create([1.0, 2.0, 3.0]);  
var v02 = vec3.create([1.0, 2.0, -2.0]);  
vec3.direction(v01, v02);  
    // 点 v01 から点 v02 に向かう方向の単位ベクトルが v01 に入る。
```

二つの点を結ぶ単位方向ベクトル (b)

glMatrix_exercise_03.html

```
var v01 = vec3.create([1.0, 2.0, 3.0]);  
var v02 = vec3.create([1.0, 2.0, -2.0]);  
var v03 = vec3.create();  
vec3.direction(v01, v02, v03);  
    // 点 v01 から点 v02 に向かう方向の単位ベクトルが v03 に入る。  
    // v01 は変わらず。
```

二点間の線形補間 (a)

glMatrix_exercise_03.html

```
var v01 = vec3.create([1.0, 2.0, 3.0]);
var v02 = vec3.create([1.0, 2.0, -2.0]);
var s = 0.6;
vec3.lerp(v01, v02, s);
// 点 v01 と点 v02 を結ぶ線分を比率 s で線形補間した位置ベクトル
// 結果は v01 に入る。
```

二点間の線形補間 (b)

glMatrix_exercise_03.html

```
var v01 = vec3.create([1.0, 2.0, 3.0]);  
var v02 = vec3.create([1.0, 2.0, -2.0]);  
var v03 = vec3.create();  
var s = 0.6;  
vec3.lerp(v01, v02, s, v03);
```

// 点 v01 と点 v02 を結ぶ線分を比率 s で線形補間した位置ベクトル
// 結果は v03 に入る。v01 は変わらず。

4x4 単位行列

glMatrix_exercise_04.html

単位行列（もっと便利な作り方は次のページ）

```
var m01 = mat4.create([
  1.0, 0.0, 0.0, 0.0, // 1st column
  0.0, 1.0, 0.0, 0.0, // 2nd column
  0.0, 0.0, 1.0, 0.0, // 3rd column
  0.0, 0.0, 0.0, 1.0 // 4th column
]);
```

4x4 単位単位行列

glMatrix_exercise_04.html

単位行列を作るにはこうするのが簡単

```
var m01 = mat4.create();  
mat4.identity(m01);
```

転置 (a)

glMatrix_exercise_04.html

```
var m01 = mat4.create([
    1.0, 1.0, 1.0, 1.0, // 1st column
    0.0, 1.0, 1.0, 1.0, // 2nd column
    0.0, 0.0, 1.0, 1.0, // 3rd column
    0.0, 0.0, 0.0, 1.0 // 4th column
]);
mat4.transpose(m01);
// m01 の転置
```

転置 (b)

glMatrix_exercise_04.html

```
var m01 = mat4.create([
    1.0, 1.0, 1.0, 1.0, // 1st column
    0.0, 1.0, 1.0, 1.0, // 2nd column
    0.0, 0.0, 1.0, 1.0, // 3rd column
    0.0, 0.0, 0.0, 1.0 // 4th column
]);
var m02 = mat4.create();
mat4.transpose(m01,m02);
// m01 の転置が m02 に入る。m01 は変わらず。
```

行列

行列成分の配置は列優先であることに注意

```
mat4.create([  
    a00, a10, a20, a30, // 第1列  
    a01, a11, a21, a31, // 第2列  
    a02, a12, a22, a32, // 第3列  
    a03, a13, a23, a33 // 第4列  
]);
```

行列のかけ算

```
var m03 = mat4.multiply(m01, m02)
```

行列 m01 と行列 m02 をかけた結果が m03 に入る。

行列 m01 も書き換わる (=m03)

行列 m01 を変更したくない場合は

```
mat4.multiply(m01, m02, m03)
```

とする。

行列式

glmatrix_exercise_04.html

```
var m01 = mat4.create([
    1.0, 1.0, 2.0, 1.0, // 1st column
    2.0, 1.0, 2.0, 2.0, // 2nd column
    0.0, 2.0, 3.0, 3.0, // 3rd column
    0.0, 3.0, 4.0, 4.0 // 4th column
]);
console.log(" det(m01) = " + mat4.determinant(m01)); // = -2
```

逆行列 (a)

glMatrix_exercise_04.html

```
var m01 = mat4.create([
    1.0, 1.0, 2.0, 1.0, // 1st column
    2.0, 1.0, 2.0, 2.0, // 2nd column
    0.0, 2.0, 3.0, 3.0, // 3rd column
    0.0, 3.0, 4.0, 4.0 // 4th column
]);
mat4.inverse(m01);
```

m01 が自分の逆行列に置き換わる

逆行列 (b)

glMatrix_exercise_04.html

```
var m01 = mat4.create([
    1.0, 1.0, 2.0, 1.0, // 1st column
    2.0, 1.0, 2.0, 2.0, // 2nd column
    0.0, 2.0, 3.0, 3.0, // 3rd column
    0.0, 3.0, 4.0, 4.0 // 4th column
]);
var m02 = mat4.inverse(m01);
```

m01 が自分の逆行列に置き換わる。m02 に同じものが入る。

逆行列 (c)

glMatrix_exercise_04.html

```
var m01 = mat4.create([
    1.0, 1.0, 2.0, 1.0, // 1st column
    2.0, 1.0, 2.0, 2.0, // 2nd column
    0.0, 2.0, 3.0, 3.0, // 3rd column
    0.0, 3.0, 4.0, 4.0 // 4th column
]);
var m02 = mat4.create();
mat4.inverse(m01,m02); //m01 が自分の逆になる。m02 も同じもの
var m03 = mat4.create(); mat4.multiply(m01,m02,m03) // m01 は不変
```

3x3 から 4x4 行列へ (a)

glMatrix_exercise_05.html

```
var m01 = mat3.create([
    1.0, 2.0, 3.0, // 1st column
    0.0, 1.0, 2.0, // 2nd column
    0.0, 0.0, 1.0 // 3rd column
]);
var m02 = mat3.toMat4(m01);
```

3x3 から 4x4 行列へ (b)

glMatrix_exercise_05.html

左上の 3 行 3 列に要素をコピーする。それ以外は、4 行 4 列目は 1、それ以外は 0 でうめる。

```
var m01 = mat3.create([
    1.0, 2.0, 3.0, // 1st column
    0.0, 1.0, 2.0, // 2nd column
    0.0, 0.0, 1.0 // 3rd column
]);
var m02 = mat4.create();
mat3.toMat4(m01,m02)
// m01 = [1, 2, 3, 0, 1, 2, 0, 0, 1]
// m02 = [1, 2, 3, 0, 0, 1, 2, 0, 0, 0, 1, 0, 0, 0, 0, 1]
```

4x4 から 3x3 行列へ (a)

glMatrix_exercise_05.html

左上の 3x3 成分をとりだす。

```
var m01 = mat4.create([
    1.0, 1.0, 2.0, 1.0, // 1st column
    2.0, 1.0, 2.0, 2.0, // 2nd column
    0.0, 2.0, 3.0, 3.0, // 3rd column
    0.0, 3.0, 4.0, 4.0 // 4th column
]);
var m02 = mat4.toMat3(m01);

// m01 = [1, 1, 2, 1, 2, 1, 2, 2, 0, 2, 3, 3, 0, 3, 4, 4]
// m02 = [1, 1, 2, 2, 1, 2, 0, 2, 3]
```

4x4 から 3x3 行列へ (b)

glMatrix_exercise_05.html

左上の 3x3 成分をとりだす。

```
var m01 = mat4.create([
    1.0, 1.0, 2.0, 1.0, // 1st column
    2.0, 1.0, 2.0, 2.0, // 2nd column
    0.0, 2.0, 3.0, 3.0, // 3rd column
    0.0, 3.0, 4.0, 4.0 // 4th column
]);
var m02 = mat4.create();
mat4.toMat3(m01, m02);

// m01 = [1, 1, 2, 1, 2, 1, 2, 2, 0, 2, 3, 3, 0, 3, 4, 4]
// m02 = [1, 1, 2, 2, 1, 2, 0, 2, 3]
```

3x3 行列のコピー

`mat3.set()`

使い方は `vec3.set()` と同じ

これまでの説明で以下の引数入出力パターンには十分なじんだと思うので、以後、並記はしない。

```
mat4.operator(src_and_dest, param)
```

```
dest = mat4.operator(src_and_dest, param)
```

```
mat4.operator(src, param, dest)
```

ただし、次の multiplyVec3 は

```
mat4.multiplyVec3(param, src_and_dest);
```

```
mat4.multiplyVec3(param, src, dest);
```

というパターンである。

4x4 行列と vec3 の積 (a)

glMatrix_exercise_05.html

```
var M = mat4.create([
  1.0, 0.0, 0.0, 0.0,
  2.0, 0.0, 1.0, 0.0,
  3.0, 1.0, 0.0, 0.0,
  4.0, 0.0, 0.0, 1.0
]);
var v01 = vec3.create([1, 0, 0]);
  // [1, 0, 0, 1] という vec4 に変換されてから、積をとる
mat4.multiplyVec3(M,v01); //積の結果の vec4 の第 4 成分は捨てられる
  // v01 = [5, 0, 0]
```

4x4 行列と vec3 の積 (b)

glMatrix_exercise_05.html

```
var m01 = mat4.create();  
mat4.identity(m01);  
var v01 = vec3.create([1.0, 0.0, 0.0]);  
var v02 = vec3.create();  
mat4.multiplyVec3(m01, v01, v02);
```

z軸の周りの回転行列を右からかける glMatrix_exercise_05.html

```
var M = mat4.create([
    1.0, 0.0, 0.0, 0.0, // rotate
    0.0, 0.0, 1.0, 0.0, // around
    0.0,-1.0, 0.0, 0.0, // x-axis
    0.0, 0.0, 0.0, 1.0 // for 90 deg.
]);
var angle = Math.PI/2;
var N = mat4.create();
mat4.rotateZ(M, angle, N); // N = MR (not RM)
var v01 = vec3.create([1,0,0]);
mat4.multiplyVec3(N, v01); // v01 = [0, 0, 1]
```

同様な関数：

`mat4.rotateX()`：x 軸周りの回転行列をかける

`mat4.rotateY()`：y 軸周りの回転行列をかける

`mat4.rotate()`：指定した軸の周りの回転行列をかける

平行移動行列を右からかける glMatrix_exercise_05.html

```
var M = mat4.create([
    1.0, 0.0, 0.0, 0.0, // rotate
    0.0, 0.0, 1.0, 0.0, // around
    0.0,-1.0, 0.0, 0.0, // x-axis
    0.0, 0.0, 0.0, 1.0 // for 90 deg.
]);
var transv = vec3.create([0,1,0]); // translate in y
var N = mat4.create();
mat4.translate(M, transv, N); // N = MT
var v01 = vec3.create([1,0,0]);
mat4.multiplyVec3(N, v01); // v01 = [1, 0, 1]
```

射影行列の作成

- モデルビュー変換と射影変換については後で説明（復習）する。
- それぞれの関数の使い方はこの後のサンプルコードで。
- 詳細はソースコード（glMatrix.js）を参照。

```
mat4.frustum(left, right, bottom, top, near, far, dest);
```

```
mat4.perspective(fovy, aspect, near, far, dest);
```

```
mat4.ortho(left, right, bottom, top, near, far, dest);
```

```
mat4.lookAt(eye, center, up, dest);
```

その他便利な関数

`mat4.toInverseMat3` : 左上 3x3 行列の逆行列を取り出す。法線ベクトルの変換に便利

`mat4.scale` : 引数の `vec3` で指定した成分でスケールする

`quat4.*` : 4 元数関係

`{vec3,vec4,mat3,mat4,quat4}.str` : 成分の書きだし

演習

演習

- glMatrix_exercise_02.html から
- glMatrix_exercise_05.html までをベースに各自自由に。

レポート課題

- glMatrix を使い、任意の計算をせよ。画像不要。
- **アーカイブにはしない。**
- 提出はメールで。添付ファイルは2つ。
 1. レポートの PDF ファイル： **ファイル名： report_03.pdf**
 2. 作成した HTML ファイル： **ファイル名： report_03.html**
 - ファイル名中のアンダースコア (_) は半角
 - ファイル拡張子は html とし、 htm としない
- gmail アドレス： kageyama.lecture@gmail.com
- メールのタイトル： **情報可視化論 レポート 3**
- レポートの PDF ファイルは1ページのみ。(表紙をつけない。)
- レポートの PDF ファイルには以下の4点を記述する。
- レポート本文には以下を記述すること
 - 1 学籍番号と氏名
 - 2 ウェブ公開時の名前 (本名またはイニシャル)
 - 3 公開用タイトル (20字以内)
例：「 4×4 の直交行列とその転置行列の積」
 - 4 glMatrix を使ってどのような計算をしたか
- 締め切り： 6/23 (火) 午前 7:00