システム工学概論

6. Cによる数値計算と可視化（発展編）

山田泰司

taiji@aihara.co.jp

株式会社あいはら 研究開発チーム

（例題１）数値解析とｎ次元グラフィックス

以下のハイパーレスラー方程式と呼ばれる常微分方程式：

$$\begin{aligned}
\frac{dx}{dt} &= -y - z \\
\frac{dy}{dt} &= x + ay + w \\
\frac{dz}{dt} &= b + xz \\
\frac{dw}{dt} &= cw - dz
\end{aligned}$$

において $a = 0.25, b = 2.2, c = 0.05, c = 0.5, x_0 = -19, y_0 = 0,$ $z_0 = 0, w_0 = 0$ **における** $t < 20000\delta t$ **(例えば、** $\delta t = 0.01$**) までの値をディスプレイにて可視化せよ。**

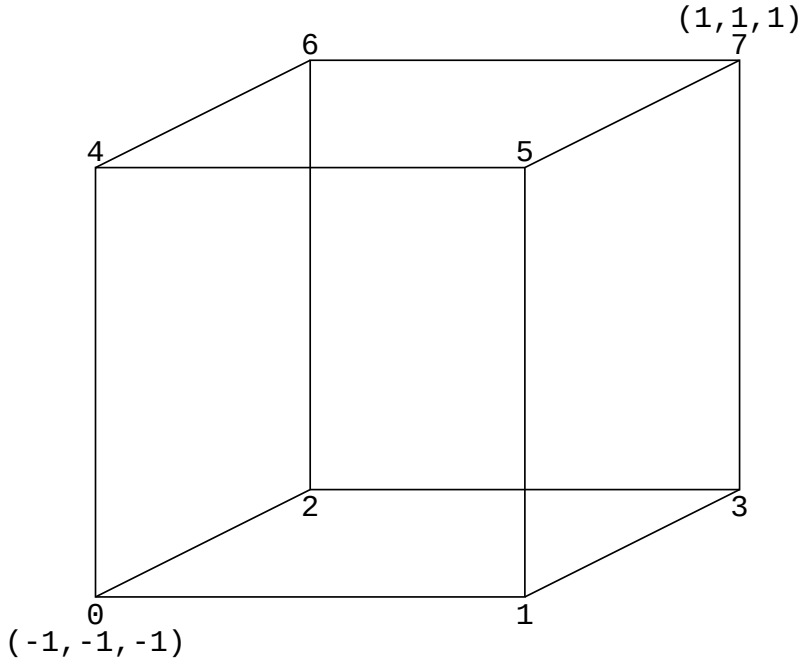
（例題０）数値解析と３次元グラフィックス

以下のレスラー方程式と呼ばれる常微分方程式：

$$\begin{aligned}\frac{dx}{dt} &= -y - z \\ \frac{dy}{dt} &= x + ay \\ \frac{dz}{dt} &= b + z(x - c)\end{aligned}$$

において $a = 0.398, b = 2, c = 4, x_0 = 0.1, y_0 = 0, z_0 = 0$ **における** $t < 20000\delta t$ **(例えば、** $\delta t = 0.01$**) までの値をディスプレイにて可視化せよ。**

立方体の頂点と線分



線分：

(0,4), (2,6), (1,5), (3,7),
(0,2), (4,6), (1,3), (5,7),
(0,1), (4,5), (2,3), (6,7),

立方体の頂点と線分（続き）

立方体の頂点と線分を渡すルーチン：

```
void cube3(double lb[3], double ub[3],
           double vertices[8][3], int segments[12*2])
{
  int i, j;
  static int segs[12*2] = { /* 線分の数: 12 */
    0,4, 2,6, 1,5, 3,7,
    0,2, 4,6, 1,3, 5,7,
    0,1, 4,5, 2,3, 6,7,
  };
  /* for (i=0; i<12*2; i++) segments[i] = segs[i]; */
  for (i=0; i<3; i++)
    for (j=0; j<12/3; j++) {
      segments[(12/3*i+j)*2+0] = segs[(12/3*i+j)*2+0];
      segments[(12/3*i+j)*2+1] = segs[(12/3*i+j)*2+1];
    }
  for (i=0; i<8; i++) { /* 頂点の数: 8 */
    vertices[i][0] = ((i>>0)%2) ? ub[0] : lb[0];
    vertices[i][1] = ((i>>1)%2) ? ub[1] : lb[1];
    vertices[i][2] = ((i>>2)%2) ? ub[2] : lb[2];
  }
}
```

n次元グラフィックスの手段

３次元グラフィックスの手段：

- X11 **等の２次元座標系へ（自前で）３次元空間を投影する**
- OpenGL **等の３次元グラフィックス技術を利用する**

OpenGL **プログラミングの選択肢**

- GLUT — The OpenGL Utility Toolkit
- OpenGL + X11 — glx
- OpenGL + SDL
- **その他、各プラットフォームのウィンドウィングに沿った形態**

等の３次元空間へ（自前で）ｎ次元空間を投影！

３次元空間における回転

X 軸まわり角度 θ の回転の変換行列：

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}$$

Y 軸まわり角度 θ の回転の変換行列：

$$R_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

Z 軸まわり角度 θ の回転の変換行列：

$$R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

（使用例）３次元ベクトル $P \to P'$ の変換：

$$P' = R_x(\theta_x)R_y(\theta_y)R_z(\theta_z)P$$

３次元空間における回転（続き）

```
void xrot_mat3(double a, double m[3][3])
{ /* X 軸まわりの回転の変換行列 */
  double c = cos(a), s = sin(a);

  m[0][0] = 1; m[0][1] = 0; m[0][2] =  0;
  m[1][0] = 0; m[1][1] = c; m[1][2] = -s;
  m[2][0] = 0; m[2][1] = s; m[2][2] =  c;
}
void yrot_mat3(double a, double m[3][3])
{ /* Y 軸まわりの回転の変換行列 */
  double c = cos(a), s = sin(a);

  m[0][0] =  c; m[0][1] = 0; m[0][2] = s;
  m[1][0] =  0; m[1][1] = 1; m[1][2] = 0;
  m[2][0] = -s; m[2][1] = 0; m[2][2] = c;
}
void zrot_mat3(double a, double m[3][3])
{ /* Z 軸まわりの回転の変換行列 */
  double c = cos(a), s = sin(a);

  m[0][0] = c; m[0][1] = -s; m[0][2] = 0;
  m[1][0] = s; m[1][1] =  c; m[1][2] = 0;
  m[2][0] = 0; m[2][1] =  0; m[2][2] = 1;
}
```

３次元空間における回転（続き）

```
void mat3_mult_mat3(double a[3][3], double b[3][3], double c[3][3])
{ /* 行列×行列 */
  int i, j, k;

  for (i=0; i<3; i++)
    for (j=0; j<3; j++)
      for (c[i][j] = 0, k=0; k<3; k++)
        c[i][j] += a[i][k]*b[k][j];
}
void mat3_mult_vec3(double a[3][3], double b[3], double c[3])
{ /* 行列×ベクトル */
  int i, j;

  for (i=0; i<3; i++)
    for (c[i] = 0, j=0; j<3; j++)
      c[i] += a[i][j]*b[j];
}
        :
  double mat3[3][3];
  double matx[3][3], maty[3][3], matz[3][3], matw[3][3];

  xrot_mat3(deg2rad(v->deg[0]), matx);
  yrot_mat3(deg2rad(v->deg[1]), maty);
  zrot_mat3(deg2rad(v->deg[2]), matz);
  mat3_mult_mat3(matx, maty, matw);
  mat3_mult_mat3(matw, matz, mat3); /* 回転の変換行列を求める */
        :
  mat3_mult_vec3(mat3, p, r); /* 点を変換する */
```

３次元空間における四元数による回転

四元数（クォータニオン）：

$$q = (w, x, y, z) = w + xi + yj + zk = s + v, \qquad v = (x, y, z)$$

クォータニオンの虚数成分：

$$i^2 = j^2 = k^2 = -1,$$
$$ij = -ji = k, \quad jk = -kj = i, \quad ki = -ik = j,$$

q の共役 $\bar{q}$、内積、逆元 q^{-1}：

$$\bar{q} = s - v, \quad q\bar{q} = \bar{q}q = q \cdot q = |q|^2, \quad q^{-1} = \frac{\bar{q}}{|q|^2}$$

q による３次元ベクトル p の回転：

$$R_q(p) = qpq^{-1}$$

～四元数による回転（続き）

q_1 と q_2 の乗算：

$$q_1 q_2 = s_1 s_2 - v_1 v_2 + s_1 v_2 + v_1 s_2 + v_1 \times v_2$$

q と３次元ベクトル p の乗算：

$$qp = -vp + sp + v \times p$$

回転の変換行列のオイラー角 (ヨー ψ, ピッチ τ, ロール φ)：

$$\varphi = \tan^{-1} \frac{R_{3,2}}{R_{3,3}}, \quad \tau = \sin^{-1} -R_{3,1}, \quad \psi = \tan^{-1} \frac{R_{2,1}}{R_{1,1}}$$

オイラー角 (ヨー ψ, ピッチ τ, ロール φ) の回転の q：

$$q_\varphi = (\cos(\varphi/2), \sin(\varphi/2), 0, 0),$$
$$q_\tau = (\cos(\tau/2), 0, \sin(\tau/2), 0),$$
$$q_\psi = (\cos(\psi/2), 0, 0, \sin(\psi/2)),$$
$$q = q_\psi q_\tau q_\varphi$$

～四元数による回転（続き）

```
void co_qua(double q[4], double qo[4]) /* 共役四元数 */
{
  qo[3] =   q[3];
  qo[0] = - q[0];
  qo[1] = - q[1];
  qo[2] = - q[2];
}
void qua_mult_qua(double q1[4], double q2[4], double qo[4])
{ /* 四元数×四元数 */
  qo[3] = q1[3]*q2[3] - q1[0]*q2[0] - q1[1]*q2[1] - q1[2]*q2[2];
  qo[0] = q1[3]*q2[0] + q1[0]*q2[3] + q1[1]*q2[2] - q1[2]*q2[1];
  qo[1] = q1[3]*q2[1] + q1[1]*q2[3] + q1[2]*q2[0] - q1[0]*q2[2];
  qo[2] = q1[3]*q2[2] + q1[2]*q2[3] + q1[0]*q2[1] - q1[1]*q2[0];
}
void qua_mult_vec3(double q[4], double v[3], double qo[4])
{ /* 四元数×ベクトル */
  qo[3] = - q[0]*v[0] - q[1]*v[1] - q[2]*v[2];
  qo[0] =   q[3]*v[0] + q[1]*v[2] - q[2]*v[1];
  qo[1] =   q[3]*v[1] + q[2]*v[0] - q[0]*v[2];
  qo[2] =   q[3]*v[2] + q[0]*v[1] - q[1]*v[0];
}
void ang_to_qua(double a[3], double q[4]) /* オイラー角の回転の四元数 */
        :
void mat3_to_ang(double m[3][3], double a[3]) /* 変換行列のオイラー角 */
        :
```

～四元数による回転（続き）

```
void qua_rot_vec3(double q[4], double v[3], double qo[4])
{ /* 四元数によるベクトルの回転 */
  double qv[4], coq[4];

  qua_mult_vec3(q, v, qv);
  co_qua(q, coq);
  qua_mult_qua(qv, coq, qo);
}
        :
  double matx[3][3], maty[3][3], matz[3][3], matw[3][3], ang[3];

  xrot_mat3(deg2rad(v->deg[0]), matx);
  yrot_mat3(deg2rad(v->deg[1]), maty);
  zrot_mat3(deg2rad(v->deg[2]), matz);
  mat3_mult_mat3(matx, maty, matw);
  mat3_mult_mat3(matw, matz, mat3);
  mat3_to_ang(mat3, ang);
  ang_to_qua(ang, qua); /* 回転の四元数を求める */
        :
  qua_rot_vec3(qua, p, r); /* 点を回転する */
```

～四元数による回転（続き）

```
void qua_rot_vec3(double q[4], double v[3], double qo[4])
{ /* 四元数によるベクトルの回転 */
  double qv[4], coq[4];

  qua_mult_vec3(q, v, qv);
  co_qua(q, coq);
  qua_mult_qua(qv, coq, qo);
}
        :
  double q_x[] = {0,0,0,0}, q_y[] = {0,0,0,0}, q_z[] = {0,0,0,0}, q_w[4];

  q_x[3] = cos(deg2rad(v->deg[0])/2); q_x[0] = sin(deg2rad(v->deg[0])/2);
  q_y[3] = cos(deg2rad(v->deg[1])/2); q_y[1] = sin(deg2rad(v->deg[1])/2);
  q_z[3] = cos(deg2rad(v->deg[2])/2); q_z[2] = sin(deg2rad(v->deg[2])/2);
  qua_mult_qua(q_x, q_y, q_w);
  qua_mult_qua(q_w, q_z, qua); /* 回転の四元数を求める */
        :
  qua_rot_vec3(qua, p, r); /* 点を回転する */
```

n 次元空間における回転

k 軸から l 軸への角度 θ の回転の変換行列：

$$R_{k\to l}(\theta) = \begin{cases} R_{i,j} = \cos\theta & \text{if } i = j = k \\ R_{i,j} = -\sin\theta & \text{if } i = k, j = l \\ R_{i,j} = \sin\theta & \text{if } i = l, j = k \\ R_{i,j} = \cos\theta & \text{if } i = j = l \\ R_{i,j} = 1 & \text{if } i = j \\ R_{i,j} = 0 & \text{otherwise} \end{cases}$$

（使用例） n 次元ベクトル $P \to P'$ の変換：

$$P' = \left(\prod_{i=0,j=0}^{i<n,j<n,i\neq j} R_{i\to j}(\theta_{i\to j}) \right) P$$

n 次元空間における回転（続き）

```
double **mat(int M, int N) /* 行列の確保 */
{
  int i;
  double **m;

  m = (double **)malloc(sizeof(double *)*M);
  for (i=0; i<M; i++)
    m[i] = (double *)malloc(sizeof(double)*N);
  return m;
}
void free_mat(int M, double **m) /* 行列の解放 */
{
  int i;

  for (i=0; i<M; i++)
    free(m[i]);
  free(m);
}
```

n 次元空間における回転（続き）

```
void unit_mat(int M, int N, double **m) /* 単位行列 */
{
  int i, j;

  for (i=0; i<M; i++)
    for (j=0; j<N; j++)
      m[i][j] = (i==j) ? 1 : 0;
}
        :
void rot_mat(int M, int f, int t, double a, double **m)
{ /* f 軸から t 軸への角度 a の回転の変換行列 */
  double c = cos(a), s = sin(a);

  unit_mat(M, M, m);
  m[f][f] = c; m[f][t] = -s;
  m[t][f] = s; m[t][t] =  c;
}
```

n 次元空間における回転（続き）

```
void cp_mat(int M, int N, double **a, double **b) /* 行列の複製 */
{
  int i, j;

  for (i=0; i<M; i++)
    for (j=0; j<N; j++)
      b[i][j] = a[i][j];
}
void mat_mult_mat(int M, int N, int L, double **a, double **b, double **c)
{ /* 行列×行列 */
  int i, j, k;

  for (i=0; i<M; i++)
    for (j=0; j<L; j++)
      for (c[i][j] = 0, k=0; k<N; k++)
        c[i][j] += a[i][k]*b[k][j];
}
void mat_mult_vec(int N, double **a, double b[], double c[])
{ /* 行列×ベクトル */
  int i, j;

  for (i=0; i<N; i++)
    for (c[i] = 0, j=0; j<N; j++)
      c[i] += a[i][j]*b[j];
}
```

n 次元空間における回転（続き）

```
    double **matn = mat(odes->ode->N, odes->ode->N);
           :
    int n = odes->ode->N;
    double **matm = mat(n, n), **m_ij = mat(n, n);
           :
    unit_mat(n, n, matn);
    for (i=0; i<n; i++)
      for (j=0; j<n; j++)
        if (i != j) {
          rot_mat(n, i, j, deg2rad(v->hdeg), m_ij);
          cp_mat(n, n, matn, matm);
          mat_mult_mat(n, n, n, matm, m_ij, matn);
        } /* 回転の変換行列を求める */
    free_mat(n, matm);
    free_mat(n, m_ij);
           :
    mat_mult_vec(odes->ode->N, matn, p, q); /* 点を変換する */
    qua_rot_vec3(qua, q, r); /* さらに点を変換する */
```

立方体の頂点と線分（続き）

立方体を描く：

```
{
  int s[12*2];
  double lb[3] = {-1,-1,-1}, ub[3] = {1,1,1}, p[8][3], r[8][3];

  cube3(lb, ub, p, s); /* 立方体の頂点と線分を得る */
  for (i=0; i<8; i++)
    qua_rot_vec3(qua, p[i], r[i]); /* すべての頂点を変換する */
  for (i=0; i<12; i++) {
    int i0 = s[i*2], i1 = s[i*2+1];
    x0 = (r[i0][0]*v->s + 1)*v->w/2; y0 = (-r[i0][1]*v->s + 1)*v->h/2;
    x1 = (r[i1][0]*v->s + 1)*v->w/2; y1 = (-r[i1][1]*v->s + 1)*v->h/2;
    XDrawLine(v->display, v->pixmap, v->gc, x0, y0, x1, y1);
  } /* すべての線分を描く */
}
```

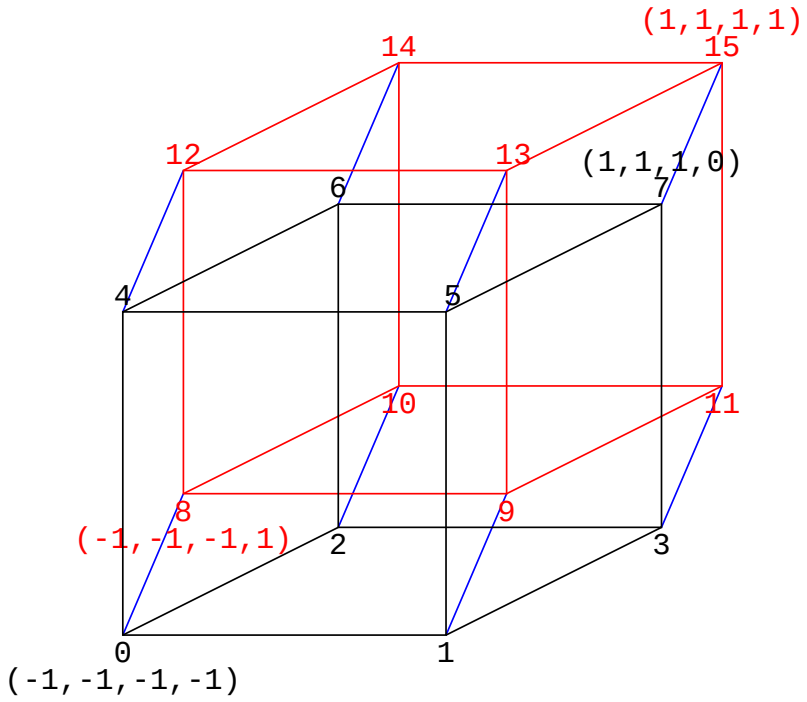
立方体の頂点と線分（続き）

立方体の頂点と線分を求めるルーチン：

```
void cube3(double lb[3], double ub[3],
           double vertices[8][3], int segments[12*2])
{
  int i, j;

  for (i=0; i<3; i++)
    for (j=0; j<12/3; j++) { /* 線分の数: 2^2*3 */
      int v0 = 0, v1 = 0, k;
      for (k=0; k<3; k++) {
        v0 = ((i>k)?((j>>k)%2):(i==k)?0:((j>>(k-1))%2)) + 2*v0;
        v1 = ((i>k)?((j>>k)%2):(i==k)?1:((j>>(k-1))%2)) + 2*v1;
      }
      segments[(12/3*i+j)*2+0] = v0;
      segments[(12/3*i+j)*2+1] = v1;
    }
  for (i=0; i<1<<3; i++) /* 頂点の数: 2^3 */
    for (j=0; j<3; j++)
      vertices[i][j] = ((i>>j)%2) ? ub[j] : lb[j];
}
```

超立方体の頂点と線分



線分：

(0,8), (4,12), (2,10), (6,14), (1,9), (5,13), (3,11), (7,15)
(0,4), (8,12), (2,6), (10,14), (1,5), (9,13), (3,7), (11,15)
(0,2), (8,10), (4,6), (12,14), (1,3), (9,11), (5,7), (13,15)
(0,1), (8,9), (4,5), (12,13), (2,3), (10,11), (6,7), (14,15)

超立方体の頂点と線分（続き）

超立方体の頂点と線分を求めるルーチン：

```
void cube(int n, double lb[], double ub[],
          double **vertices, int *segments)
{
  long i, j, nv = 1<<n, ns = (1<<(n-1))*n;

  for (i=0; i<n; i++)
    for (j=0; j<ns/n; j++) { /* 線分の数: 2^(n-1)*n */
      int v0 = 0, v1 = 0, k;
      for (k=0; k<n; k++) {
        v0 = ((i>k)?((j>>k)%2):(i==k)?0:((j>>(k-1))%2)) + 2*v0;
        v1 = ((i>k)?((j>>k)%2):(i==k)?1:((j>>(k-1))%2)) + 2*v1;
      }
      segments[(ns/n*i+j)*2+0] = v0;
      segments[(ns/n*i+j)*2+1] = v1;
    }
  for (i=0; i<nv; i++) /* 頂点の数: 2^n */
    for (j=0; j<n; j++)
      vertices[i][j] = ((i>>j)%2) ? ub[j] : lb[j];
}
```

超立方体の頂点と線分（続き）

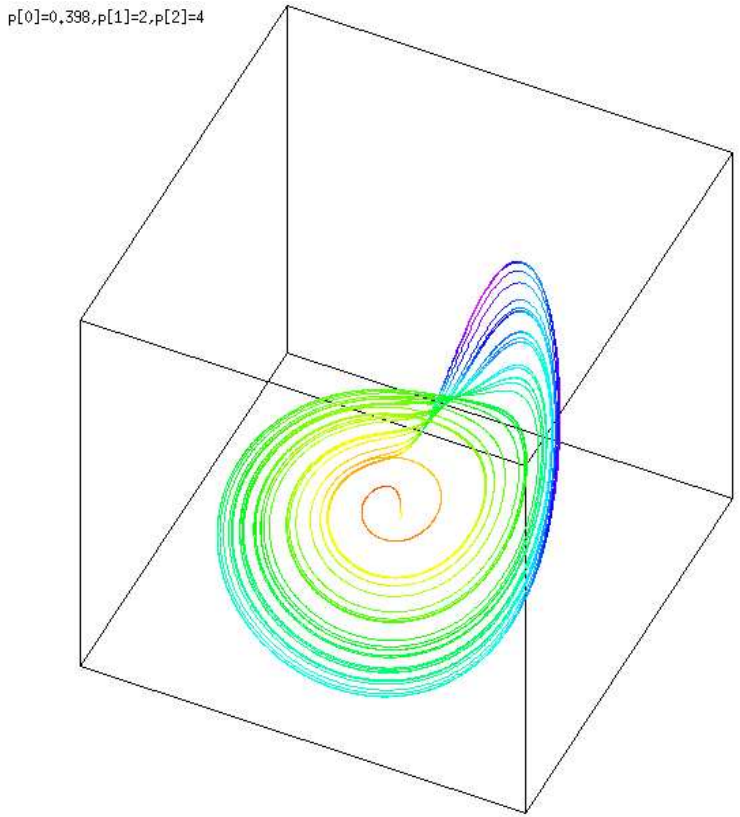
超立方体を描く：

```
{
  int n = odes->ode->N, nv = 1<<n, ns = (1<<(n-1))*n, s[ns*2];
  double lb[n], ub[n], **p = mat(nv,n), **q = mat(nv,n), **r = mat(nv,n);

  for (i=0; i<n; i++) {
    lb[i] = -1; ub[i] = 1;
  }
  cube(n, lb, ub, p, s); /* 超立方体の頂点と線分を得る */
  for (i=0; i<nv; i++)
    mat_mult_vec(n, matn, p[i], q[i]); /* すべての頂点を変換する */
  for (i=0; i<nv; i++)
    qua_rot_vec3(qua, q[i], r[i]); /* さらにすべての頂点を変換する */
  for (i=0; i<ns; i++) {
    int i0 = s[i*2], i1 = s[i*2+1];
    x0 = (r[i0][0]*v->s + 1)*v->w/2; y0 = (-r[i0][1]*v->s + 1)*v->h/2;
    x1 = (r[i1][0]*v->s + 1)*v->w/2; y1 = (-r[i1][1]*v->s + 1)*v->h/2;
    XDrawLine(v->display, v->pixmap, v->gc, x0, y0, x1, y1);
  }
  free_mat(nv, p);
  free_mat(nv, q);
  free_mat(nv, r);
}
```

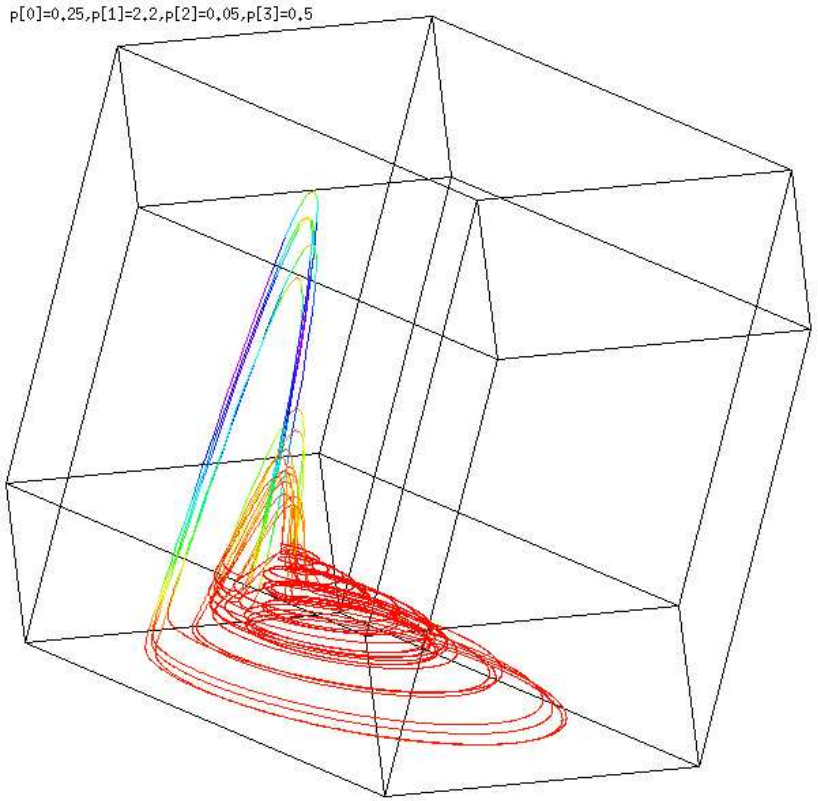
（例題０）の解答例

```
$ diff -u v_lorenz.c v_ode00.c
# 次元に注意しながらプログラム修正
$ ./v_ode00.c
```



（例題１）の解答例

```
$ diff -u v_ode00.c v_ode.c
# グラフィックの次元を一般化する
$ ./v_ode.c
```



数値計算とｎＤグラフィックス：まとめ

数値計算結果の理解や表現を助ける可視化技術 — 自らが理解を深めるには可視化された数値計算結果を積極的に観ることが重要である：

- ▷ ｎ次元空間に数値計算結果を可視化するための技術
- ▷ 加えて、時間軸を数値計算に活用するアニメーション技術
- ▷ さらに、色空間を数値計算に活用する色管理技術

いろいろな物理計算

- 運動力学（剛体、ラグランジュ運動方程式、衝突、多体問題など）
- 流体力学（ナビエ・ストークス方程式など）
- 機械工学（航空工学、船舶工学、自動車工学、ロボット工学、宇宙工学など）
- 複雑系工学、非線形力学、カオス理論など

ゲームと物理計算：

- 可視判定、衝突判定・応答などから、
- 運動力学シミュレーション、剛体振り子、ロボット
- 流体シミュレーション、連続体シミュレーション
- 高次元力学系シミュレーション、多体問題、粒子系シミュレーションなどなど

剛体振り子、衝突、多体問題、高次元モデルについては（手前味噌ですが）：

`http://www.aihara.co.jp/~taiji/pendula-equations/present.html`
`http://www.aihara.co.jp/~taiji/lecture-2007/win32/`

- さまざまな振り子のシミュレータ
- 惑星の多体問題のシミュレータ
- ビリアードモデルのシミュレータ
- オートマトンのシミュレータ、があります。よかったら遊んでみて下さい。