

# Javaによる波動方程式の数値解析

4 年 1 6 組 5 2 番

三井康之

2002 年 3 月 22 日

# はじめに

はじめにこの卒論の目的を述べておこう。目的をまとめるなら

- 世界中の人に波動方程式の数値シミュレーションを体感してもらう

の一つになるのだが、これをするための目的として

- J a v a で波動方程式の数値解析のプログラムを作る。
- H P に載せるためにG U I を工夫する。
- J a v a で3次元のグラフを描くためのプログラムを作る。

なぜJ a v a でプログラムを作るか？J a v a には「プラットフォームに依存しない」という大きな特徴がある。一度プログラムを作ってしまうと、様々なハードウェア・ソフトウェア環境で実行することが可能である。元来全ての人に同じ環境を与えることが非常に困難であったが、J a v a ではW e b ブラウザなどで実行することによってこの点を解決できる。また、タブレットを用いることにより、より多くの方々に数値解析を容易に体感していただけるのである。

しかしながらJ a v a には3次元で数値解析をするための便利なツールがまだそろっていないのである。そこで私自身がそのツールを作らなければいけなかったのである。

この卒論では上の3つのことに力を入れている。そして実際にJ a v a でプログラミングを作り、そのプログラミングの解説をしている。

最後にこの卒業研究を進めるにあたって丁寧なご指導と惜しみない援助をしてくださった明治大学数学科助教授の桂田祐史先生に深く感謝いたします。

また私用でこの卒業研究を最後までやり遂げられなかったことに対して少し後悔しています。

# 目 次

第 1 章	1 次元波動方程式	4
1.1	差分解	4
1.2	境界条件	6
1.2.1	Dirichlet 境界条件	6
1.2.2	Neumann 境界条件の素朴な差分近似	6
1.2.3	Neumann 境界条件の仮想格子点を用いる差分近似	6
1.3	Wave1d program	7
第 2 章	2 次元波動方程式	12
2.1	差分解	12
2.2	境界条件	14
2.2.1	Dirichlet 境界条件	14
2.2.2	Neumann 境界条件の素朴な差分近似	14
2.2.3	Neumann 境界条件の仮想格子点を用いた差分近似	14
2.3	Wave2d program	18
第 3 章	Java によるアニメーション	24
3.1	WaveAll program	24
3.2	プログラムの解説	45
3.2.1	アプレット	45
3.2.2	マルチスレッドプログラミング	46
3.2.3	ダブルバッファリング	47
3.2.4	各メソッド	47
第 4 章	MitsuiWorld のドキュメンテーション	54
4.1	MitsuiWorld の一覧表	55
4.2	座標系のお話	55
4.3	MitsuiWorld の使い方	56
4.3.1	2 次元 3 次元共通のメソッド	57
4.3.2	2 次元に関するメソッド	58
4.3.3	3 次元に関するメソッド	60
4.4	MitsuiWorld のアルゴリズム	68
4.4.1	2 次元の座標変換	68
4.4.2	3 次元の座標の動き	69
4.4.3	透視投影	70
4.4.4	3 次元の座標変換	70

4.4.5	BirdView メソッド . . . . .	71
4.4.6	Zバッファによる陰面消去法 . . . . .	73
4.5	MitsuiWorld program . . . . .	74
付 録 A	3次元波動方程式のプログラム . . . . .	83

# 第1章 1次元波動方程式

## 1.1 差分解

波動方程式

$$\frac{1}{c^2}u_{tt} = u_{xx} \quad (t > 0, x \in (0, 1)) \quad (1.1)$$

と初期条件

$$u(x, 0) = \phi(x) \quad (1.2)$$

$$u_t(x, 0) = \psi(x) \quad (x \in [0, 1]) \quad (1.3)$$

それといづれかの境界条件

$$(\text{Dirichlet 境界条件}) \quad u(0, t) = u(1, t) = 0 \quad (t > 0) \quad (1.4)$$

$$(\text{Neumann 境界条件}) \quad u_x(0, t) = u_x(1, t) = 0 \quad (t > 0) \quad (1.5)$$

からなる初期値境界値問題に対する差分方程式を求める。

まず、区間  $[0, 1]$  を  $N$  等分し、格子点を  $x_i$  と表す。すなわち

$$h = \frac{1}{N}$$

として

$$x_i = ih \quad (i = 0, 1, 2, \dots)$$

一方、 $\tau > 0$  を固定して、

$$t_j = j\tau \quad (j = 0, 1, 2, \dots)$$

とおく。格子点  $(x_i, t_j)$  において偏導関数  $u_{tt}, u_{xx}$  をそれぞれ2階中心差分近似すると、

$$u_{tt}(x_i, t_j) = \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\tau^2} + O(\tau^2)$$

$$u_{xx}(x_i, t_j) = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + O(h^2)$$

であることから、式 (1.1) より

$$\frac{1}{c^2} \frac{U_i^{j+1} - 2U_i^j + U_i^{j-1}}{\tau^2} = \frac{U_{i+1}^j - 2U_i^j + U_{i-1}^j}{h^2}$$

なる差分方程式を得る。ただし  $u_{i,j} = u(x_i, t_j)$  である。

両辺に  $\tau^2$  をかけてから移項すると

$$U_i^{j+1} = 2(1 - \lambda^2)U_i^j + \lambda^2(U_{i+1}^j + U_{i-1}^j) - U_i^{j-1} \quad \cdots \dagger$$

$$(i = 1, 2, \dots, N-1; j = 1, 2, \dots)$$

ただし、 $\lambda = \frac{c\tau}{h}$  とした。  
初期条件からは

$$U_i^0 = \phi(x_i) \quad \cdots \dagger$$

を得る。

また、 $u(x, t)$  の滑らかさを  $C^3$  級とすると、

$$u(x_i, t_1) = u(x_i, 0) + u_t(x_i, 0)\tau + \frac{u_{tt}(x_i, 0)}{2!}\tau^2 + O(\tau^3)$$

$$= u(x_i, 0) + \psi(x_i)\tau + \frac{u_{tt}(x_i, 0)}{2!}\tau^2 + O(\tau^3)$$

$t = 0$  でも波動方程式が成り立っていると仮定すると、

$$u(x_i, t_1) = u(x_i, 0) + \psi(x_i)\tau + c^2 \frac{u_{xx}(x_i, 0)}{2!}\tau^2 + O(\tau^3)$$

$$= u(x_i, 0) + \psi(x_i)\tau + \frac{c^2}{2!} \frac{u_{i+1,0} - 2u_{i,0} + u_{i-1,0}}{h^2}\tau^2 + O(h^2) + O(\tau^3)$$

故に

$$U_i^1 = U_i^0 + \psi(ih)\tau + \frac{c^2}{2!} \frac{U_{i+1}^0 - 2U_i^0 + U_{i-1}^0}{h^2}\tau^2$$

なる差分方程式を得る。

整理して

$$U_i^1 = (1 - \lambda^2)U_i^0 + \psi(ih)\tau + \frac{\lambda^2}{2}(U_{i+1}^0 + U_{i-1}^0) \quad \cdots \dagger$$

以上から、式 (1.1), (1.2), (1.3) に対応して、未知数列  $U_i^j; 0 \leq i \leq N, j \geq 0$  に関する方程式系

$$U_i^{j+1} = 2(1 - \lambda^2)U_i^j + \lambda^2(U_{i+1}^j + U_{i-1}^j) - U_i^{j-1} \quad (1.6)$$

$$(1 \leq i \leq N-1, j \geq 1)$$

$$U_i^0 = \phi(x_i) \quad (0 \leq i \leq N) \quad (1.7)$$

$$U_i^1 = (1 - \lambda^2)U_i^0 + \psi(ih)\tau + \frac{\lambda^2}{2}(U_{i+1}^0 + U_{i-1}^0) \quad (1.8)$$

$$(1 \leq i \leq N-1)$$

が得られた。これらの式と境界条件により私たちは波動方程式の差分解を得ることができる。  
境界条件については次の節で述べる。

## 1.2 境界条件

### 1.2.1 Dirichlet 境界条件

境界条件は

$$u(0, t) = u(1, t) = 0 \quad (t > 0)$$

i.e.

$$U_0^j = U_1^j = 0 \quad (j = 1, 2, \dots) \quad (1.9)$$

### 1.2.2 Neumann 境界条件の素朴な差分近似

境界条件は

$$u_x(0, t) = u_x(1, t) = 0 \quad (t > 0)$$

で与えられ  $u_x(0, t)$  については前進差分近似を、 $u_x(1, t)$  については後退差分近似することを考えると、

$$\begin{aligned} u_x(0, t_j) &= u_x(x_0, t_j) = \frac{u(x_1, t_j) - u(x_0, t_j)}{h} + O(h) \\ u_x(1, t_j) &= u_x(x_N, t_j) = \frac{u(x_N, t_j) - u(x_{N-1}, t_j)}{h} + O(h) \end{aligned}$$

となり

$$\frac{U_1^j - U_0^j}{h} = \frac{U_N^j - U_{N-1}^j}{h} = 0$$

なる差分方程式を得る。

i.e.

$$U_0^j = U_1^j, \quad U_N^j = U_{N-1}^j \quad (j = 1, 2, \dots) \quad (1.10)$$

### 1.2.3 Neumann 境界条件の仮想格子点を用いる差分近似

境界条件は Neumann 境界条件と同じである。しかし格子点  $U_{-1}^j, U_{N+1}^j$  が存在するとして  $u_x(0, t), u_x(1, t)$  をともに 1 階の中心差分近似することを考えると、境界条件に対応する差分方程式として

$$U_{-1}^j = U_1^j, \quad U_{N-1}^j = U_{N+1}^j$$

を得る。

さらに式 (1.6), (1.8) が  $i = 0, i = N$  でも成り立つとすると式 (1.8) は

$$U_0^1 = (1 - \lambda^2)U_0^0 + \psi(ih)\tau + \lambda^2 U_1^0 \quad (1.11)$$

$$U_N^1 = (1 - \lambda^2)U_N^0 + \psi(ih)\tau + \lambda^2 U_{N-1}^0 \quad (1.12)$$

式 (1.6) は

$$U_0^{j+1} = 2(1 - \lambda^2)U_0^j + 2\lambda^2 U_1^j - U_0^j \quad (1.13)$$

$$U_N^{j+1} = 2(1 - \lambda^2)U_N^j + 2\lambda^2 U_{N-1}^j - U_N^j \quad (1.14)$$

となる。

## 1.3 Wave1d program

```
/** 1次元波動方程式を解くプログラム。
 * 境界条件
 * bc=0    Dirichlet 境界条件
 * bc=1    Neumann 境界条件
 * bc=2    格子点を用いる方法
 *
 * これをコンパイルするには MitsuiWorld が必要です。
 * MitsuiWorld が使える状態であれば普通のコンパイル javac Wave1d.java
 * と普通の起動方法 appletviewer Wave1d.java
 * でOK。
 * ただし appletviewer Wave1d.java としたければコメント欄のどこかに
 *
 * <applet code = Wave1d width=500 height=500></applet>
 *
 * というものが入ってないといけない。
 * それと MitsuiWorld の使い方によっては
 *
 * import Mitsui.*;
 *
 * を消すこと。
 * MitsuiWorld をパッケージとして使わない人はこれを消してください。
 */

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import Mitsui.*;           //MitsuiWorld の入っているパッケージ Mitsui を呼び出す

public class Wave1d extends Applet {
    int nfunc=0;           //これでどの関数を選んだか判断する。
    int bc=0;              //これでどの境界条件を選んだか判断する。
    MitsuiWorld_2 m = new MitsuiWorld_2();

    public void init(){
        Graphics g=getGraphics();
        m.setGraphics(g);
        m.setScreenSize(getSize().width,getSize().height);
        //[ a , b ] × [ c , d ]
        m.setArea(-0.2,1.2,-1.2,1.2);
    }

    public void paint(Graphics g){
        //0、1、-1、x、yを表示
        g.setColor(Color.red);
        g.drawString("0",55,getSize().height/2+5);
        g.drawString("1",55,45);
        g.drawString("-1",55,465);
        g.drawString("x",440,245);
        g.drawString("y",80,30);

        //選んだ関数の表示
        switch(nfunc){
            case 0:
                g.drawString(" =sin( x), =0",20,20);
                break;
            case 1:

```



```

        g.drawString("  $=0(x \leq 3/8, 8x-3(3/8 < x \leq 1/2, -8x+5(1/2 < x \leq 5/8), 0(5/8 < x), =0"$ ",20,20);
        break;
    case 2:
        g.drawString("平面波",20,20);
        break;
    case 3:
        g.drawString("平面波の合体",20,20);
        break;
}

int n = 200;
double lambda = 1.0;
double Tmax = 4.0;

double lambda2 = lambda * lambda;
double h = 1.0/n;
double tau = lambda * h;

double[] u1 = new double[n+1];
double[] u2 = new double[n+1];
double[] u3 = new double[n+1];

//初期値代入
//phi=
//psi=
for(int i=0;i<=n;i++)
    u1[i] = phi(i*h);
for(int i=1;i<=n;i++)
    u2[i] = (1-lambda2) * u1[i] +
        0.5 * lambda2 * (u1[i-1]+u1[i+1]) + tau * psi(i*h);

if(bc==0){
    u2[0]=u2[n]=0;
}
//Dirichlet 境界条件

if(bc==1){
    u2[0]=u2[1];
    u2[n]=u2[n-1];
}
//Neumann 境界条件

if(bc==2){
    //仮想格子点
    u2[0] = (1-lambda2) * u1[0] +
        0.5 * lambda2 * (2*u1[1]) + tau * psi(0.0);
    u2[n] = (1-lambda2) * u1[n] +
        0.5 * lambda2 * (2*u1[n-1]) + tau * psi(n*h);
}

//j=0 でのグラフ
m.setColor(0);
drawAxis();
m.setColor(2);
m.move(0.0,u1[0]);
for(int i=1;i<=n;i++)
    m.draw(i*h,u1[i]);

//t=0 でのグラフをけす。
m.setColor(getBackground());
m.move(0.0,u1[0]);
for(int i=1;i<=n;i++)
    m.draw(i*h,u1[i]);

```

```

//t=   でのグラフ
m.setColor(0);
drawAxis();
m.setColor(2);
m.move(0.0,u2[0]);
for(int i=1;i<=n;i++)
    m.draw(i*h,u2[i]);

int jmax = (int)Math rint(Tmax/tau);
for(int j=1;j<=jmax;j++){
    for(int i=1;i<n;i++)
        u3[i] = 2 * (1.0-lambda2) * u2[i] +
                lambda2 * (u2[i-1]+u2[i+1]) - u1[i];

    if(bc==0){ //Dirichlet 境界条件
        u3[0] = u3[n] = 0;
    }
    if(bc==1){ //Neumann 境界条件
        u3[0]=u3[1];
        u3[n]=u3[n-1];
    }
    if(bc==2){ //仮想格子点
        u3[0] = 2 * (1.0-lambda2) * u2[0] +
                lambda2 * (2*u2[1]) - u1[0];
        u3[n] = 2 * (1.0-lambda2) * u2[n] +
                lambda2 * (2*u2[n-1]) - u1[n];
    }

    //グラフを消す。
    m.setColor(getBackground());
    m.move(0.0,u2[0]);
    for(int i=1;i<=n;i++)
        m.draw(i*h,u2[i]);

    //t=   j(j>=2) のグラフを書く
    m.setColor(0);
    drawAxis();
    m.setColor(2);
    m.move(0.0,u3[0]);
    for(int i=1;i<=n;i++)
        m.draw(i*h,u3[i]);

    for(int i=0;i<n;i++)
        u1[i] = u2[i];
    for(int i=0;i<n;i++)
        u2[i] = u3[i];
}

}

//初期値 の関数
public double phi(double x){
    if(nfunc == 0){
        return Math.sin(Math.PI*x);
    }
    if(nfunc == 1){
        if(0.375<x && x<=0.5)
            return 8.0*x-3.0;
    }
}

```

```

        if(0.5<x && x<=0.625)
            return -8.0*x+5.0;
        else
            return 0.0;
    }
    if(nfunc == 2){
        if(0.0<=x && x<=0.2)
            return (Math.sin(Math.PI*(x*10-0.5))+1)/4;
        else
            return 0.0;
    }
    if(nfunc == 3){
        if(0.0<=x && x<=0.2)
            return (Math.sin(Math.PI*(x*10-0.5))+1)/4;
        if(0.9<=x && x<=1.0)
            return (Math.sin(Math.PI*((x-0.9)*20-0.5))+1)/8;
        else
            return 0.0;
    }
    else
        return 0.0;
}

//初期条件 の関数
public double psi(double x){
    if(nfunc == 0){
        return 0.0;
    }
    if(nfunc == 1){
        return 0.0;
    }
    if(nfunc == 2){
        if(0.0<=x && x<=0.2)
            return -2.5*Math.PI*Math.cos(Math.PI*(10*x-0.5));
        else
            return 0.0;
    }
    if(nfunc == 3){
        if(0.0<=x && x<=0.2)
            return -2.5*Math.PI*Math.cos(Math.PI*(10*x-0.5));
        if(0.9<=x && x<=1.0)
            return 2.5*Math.PI*Math.cos(Math.PI*((x-0.9)*20-0.5));
        else
            return 0.0;
    }
    else
        return 0.0;
}

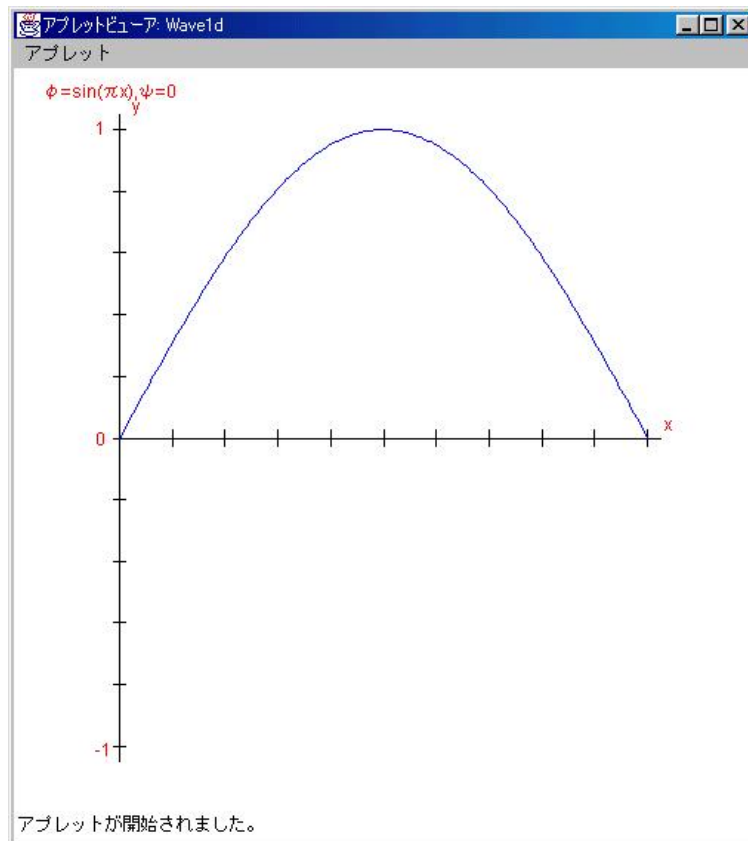
public void drawAxis(){
    //座標軸等を設定
    m.move(0.0,0.0); m.draw(1.025,0.0);
    for(double x=0.0;x<=1.0;x+=0.1){
        m.move(x,0.025); m.draw(x,-0.025);
    }
    m.move(0.0,1.05); m.draw(0.0,-1.05);
}

```

```

    for(double y=1.0;y>=-1;y-=0.2){
        m.move(-0.01,y); m.draw(0.01,y);
    }
}

```



## 第2章 2次元波動方程式

### 2.1 差分解

波動方程式

$$\begin{aligned}\frac{1}{c^2}u_{tt} &= u_{xx} + u_{yy} \quad (t > 0, (x, y) \in \Omega) \\ \Omega &= \{A < x < B, C < y < D\}\end{aligned}\tag{2.1}$$

と初期条件

$$u(x, y, 0) = \phi(x, y)\tag{2.2}$$

$$u_t(x, y, 0) = \psi(x, y) \quad ((x, y) \in \bar{\Omega})\tag{2.3}$$

と次の境界条件のいずれか

$$(\text{Dirichlet 境界条件}) \quad u(x, y, t)|_{\partial\Omega} = 0 \quad (t > 0)\tag{2.4}$$

$$(\text{Neumann 境界条件}) \quad \begin{cases} u_x(x, y, t)|_{\partial\Omega} = 0 & (t > 0) \\ u_y(x, y, t)|_{\partial\Omega} = 0 & (t > 0) \end{cases}\tag{2.5}$$

からなる初期値境界値問題に対する差分方程式を求める。

まず領域  $\Omega$  を座標軸に平行な格子点で  $x$  軸、 $y$  軸方向にそれぞれ  $N_x, N_y$  等分し、格子点をそれぞれ  $x_i, y_j$  と表す。すなわち

$$\begin{aligned}d_x &= \frac{B - A}{N_x} \\ d_y &= \frac{D - C}{N_y}\end{aligned}$$

として

$$x_i = A + id_x \quad (i = 0, 1, 2, \dots, N_x)$$

$$y_j = C + jd_y \quad (j = 0, 1, 2, \dots, N_y)$$

と置く。次に  $\tau > 0$  を固定して

$$t_k = k\tau \quad (k = 0, 1, 2, \dots)$$

と置く。格子点  $(x_i, y_j, t_k)$  において偏導関数  $u_{tt}, u_{xx}, u_{yy}$  をそれぞれ2階中心差分近似すると

$$u_{tt}(x_i, y_j, t_k) = \frac{u_{i,j,k+1} - 2u_{i,j,k} + u_{i,j,k-1}}{\tau^2} + O(\tau^2)$$

$$u_{xx}(x_i, y_j, t_k) = \frac{u_{i+1,j,k} - 2u_{i,j,k} + u_{i-1,j,k}}{d_x^2} + O(d_x^2)$$

$$u_{yy}(x_i, y_j, t_k) = \frac{u_{i,j+1,k} - 2u_{i,j,k} + u_{i,j-1,k}}{d_y^2} + O(d_y^2)$$

となり、式 (2.1) は

$$\frac{1}{c^2} \frac{U_{i,j}^{k+1} - 2U_{i,j}^k + U_{i,j}^{k-1}}{\tau^2} = \frac{U_{i+1,j}^k - 2U_{i,j}^k + U_{i-1,j}^k}{d_x^2} + \frac{U_{i,j+1}^k - 2U_{i,j}^k + U_{i,j-1}^k}{d_y^2}$$

となる。ただし  $u_{ijk} = u(x_i, y_j, t_k)$  とする。

両辺に  $\tau^2$  をかけてから移項すると、

$$U_{i,j}^{k+1} = 2(1 - \lambda_x^2 - \lambda_y^2)U_{i,j}^k + \lambda_x^2(U_{i+1,j}^k + U_{i-1,j}^k) + \lambda_y^2(U_{i,j+1}^k + U_{i,j-1}^k) - U_{i,j}^{k-1} \quad \cdots \dagger$$

$$(i = 1, 2, \dots, N_x - 1; j = 1, 2, \dots, N_y - 1; k = 1, 2, \dots)$$

が導かれる。

また初期条件より

$$U_{i,j}^0 = \phi(x_i, y_j) \quad \cdots \dagger$$

$u(x, y, t)$  の滑らかさを  $C^3$  級と仮定し、 $u(x_i, y_j, t)$  を  $t = 0$  を中心として Taylor 展開すると

$$\begin{aligned} u(x_i, y_j, t_1) &= u(x_i, y_j, 0) + u_t(x_i, y_j, 0)\tau + \frac{u_{tt}(x_i, y_j, 0)}{2!}\tau^2 + O(\tau^3) \\ &= u(x_i, y_j, 0) + \psi(x_i, y_j)\tau + \frac{u_{tt}(x_i, y_j, 0)}{2!}\tau^2 + O(\tau^3) \end{aligned}$$

$t = 0$  でも式 (2.1) が成り立つとすると

$$\begin{aligned} u(x_i, y_j, t_1) &= u(x_i, y_j, 0) + \psi(x_i, y_j)\tau + c^2 \left( \frac{u_{xx}(x_i, y_j, 0)}{2!} + \frac{u_{yy}(x_i, y_j, 0)}{2!} \right) \tau^2 + O(\tau^3) \\ &= u(x_i, y_j, 0) + \psi(x_i, y_j)\tau \\ &\quad + \frac{c^2}{2!} \left( \frac{u_{i+1,j,0} - 2u_{i,j,0} + u_{i-1,j,0}}{d_x^2} + \frac{u_{i,j+1,0} - 2u_{i,j,0} + u_{i,j-1,0}}{d_y^2} \right) \tau^2 \\ &\quad + O(d_x^2) + O(d_y^2) + O(\tau^3) \end{aligned}$$

整頓すると

$$U_{i,j}^1 = U_{i,j}^0 + \psi(A + id_x, C + jd_y)\tau + \frac{c^2}{2!} \frac{U_{i+1,j}^0 - 2U_{i,j}^0 + U_{i-1,j}^0}{d_x^2} \tau^2 + \frac{c^2}{2!} \frac{U_{i,j+1}^0 - 2U_{i,j}^0 + U_{i,j-1}^0}{d_y^2} \tau^2$$

よって

$$U_{i,j}^1 = (1 - \lambda_x^2 - \lambda_y^2)U_{i,j}^0 + \psi(A + id_x, C + jd_y)\tau + \frac{\lambda_x^2}{2}(U_{i+1,j}^0 + U_{i-1,j}^0) + \frac{\lambda_y^2}{2}(U_{i,j+1}^0 + U_{i,j-1}^0) \quad \cdots \dagger$$

以上式 (2.1), (2.2), (2.3) に対応して、

未知数列  $\{U_{i,j}^k; 0 \leq i \leq N_x, 0 \leq j \leq N_y, k = 1, 2, \dots\}$  に関する方程式系

$$U_{i,j}^{k+1} = 2(1 - \lambda_x^2 - \lambda_y^2)U_{i,j}^k + \lambda_x^2(U_{i+1,j}^k + U_{i-1,j}^k) + \lambda_y^2(U_{i,j+1}^k + U_{i,j-1}^k) - U_{i,j}^{k-1} \quad (i = 1, 2, \dots, N_x - 1; j = 1, 2, \dots, N_y - 1; k = 1, 2, \dots) \quad (2.6)$$

$$U_{i,j}^0 = \phi(x_i, y_j) \quad (0 \leq i \leq N_x, 0 \leq j \leq N_y) \quad (2.7)$$

$$U_{i,j}^1 = (1 - \lambda_x^2 - \lambda_y^2)U_{i,j}^0 + \psi(A + id_x, C + jd_y)\tau + \frac{\lambda_x^2}{2}(U_{i+1,j}^0 + U_{i-1,j}^0) + \frac{\lambda_y^2}{2}(U_{i,j+1}^0 + U_{i,j-1}^0) \quad (0 \leq i \leq N_x - 1, 0 \leq j \leq N_y - 1) \quad (2.8)$$

が得られた。これらの式と境界条件により私たちは波動方程式の差分解を得ることができる。  
境界条件は次の節で述べる。

## 2.2 境界条件

### 2.2.1 Dirichlet 境界条件

境界条件は

$$u(x, y, t)|_{\partial\Omega} = 0 \quad (t > 0)$$

i.e.

$$U_{i,j}^k = 0 \quad (i = 0 \text{ or } i = N_x \text{ or } j = 0 \text{ or } j = N_y; k = 1, 2, \dots)$$

言い換えると

$$U_{0,j}^k = U_{N_x,j}^k = 0 \quad (j = 0, 1, 2, \dots, N_y, k = 1, 2, \dots) \quad (2.9)$$

$$U_{i,0}^k = U_{i,N_y}^k = 0 \quad (i = 0, 1, 2, \dots, N_x, k = 1, 2, \dots) \quad (2.10)$$

### 2.2.2 Neumann 境界条件の素朴な差分近似

境界条件は

$$u_x(x, y, t)|_{\partial\Omega} = 0 \quad (t > 0)$$

$$u_y(x, y, t)|_{\partial\Omega} = 0 \quad (t > 0)$$

で与えられ  $u_x(x_0, y_j, t_k)$ ,  $u_y(x_i, y_0, t_k)$  については前進差分近似を、 $u_x(x_{N_x}, y_j, t_k)$ ,  $u_y(x_i, y_{N_y}, t_k)$  については後退差分近似することを考えると、

$$\begin{aligned} \frac{U_{1,j}^k - U_{0,j}^k}{d_x} &= 0, & \frac{U_{N_x,j}^k - U_{N_x-1,j}^k}{d_x} &= 0 \\ \frac{U_{i,1}^k - U_{i,0}^k}{d_y} &= 0, & \frac{U_{i,N_y}^k - U_{i,N_y-1}^k}{d_y} &= 0 \end{aligned}$$

言い換えると

$$U_{1,j}^k = U_{0,j}^k, \quad U_{N_x,j}^k = U_{N_x-1,j}^k \quad (2.11)$$

$$U_{i,1}^k = U_{i,0}^k, \quad U_{i,N_y}^k = U_{i,N_y-1}^k \quad (2.12)$$

### 2.2.3 Neumann 境界条件の仮想格子点を用いた差分近似

境界条件は Neunamm 境界条件と同じである。しかし格子点  $U_{-1,j}^k, U_{N_x+1,j}^k, U_{i,-1}^k, U_{i,N_y+1}^k$  が存在するとして  $u_x(0, y, t), u_x(N_x, y, t), u_y(x, 0, t), u_y(x, N_y, t)$  をともに 1 階の中心差分近似することを考えると、境界条件に対応する差分方程式として

$$\begin{aligned} U_{-1,j}^k &= U_{1,j}^k, & U_{N_x+1,j}^k &= U_{N_x-1,j}^k \\ U_{i,-1}^k &= U_{i,1}^k, & U_{i,N_y+1}^k &= U_{i,N_y-1}^k \end{aligned}$$

を得る。

さらに、式 (2.6), (2.8) がそれぞれ  $i = 0, i = N_x, j = 0, j = N_y$  でも成り立つとする。

(2.8) 式に、まず  $j = 0$  を代入すると

$$U_{i,0}^1 = \tau\psi(x_i, y_0) + (1 - \lambda_x^2 - \lambda_y^2)U_{i,0}^0 + \frac{1}{2}\lambda_x^2(U_{i+1,0}^0 + U_{i-1,0}^0) + \frac{1}{2}\lambda_y^2(2U_{i,1}^0)$$

となる。このだが式は  $i = 0$  で  $U_{-1,0}^0$  が  $i = N_x$  で  $U_{N_x+1,0}^0$  がでてくる。そこで  $i$  の値によって場合わけすると、

$i = 0$  のとき

$$U_{0,0}^1 = \tau\psi(x_0, y_0) + (1 - \lambda_x^2 - \lambda_y^2)U_{0,0}^0 + \frac{1}{2}\lambda_x^2(2U_{1,0}^0) + \frac{1}{2}\lambda_y^2(2U_{0,1}^0)$$

$i = 1, 2, \dots, N_x - 1$  のとき

$$U_{i,0}^1 = \tau\psi(x_i, y_0) + (1 - \lambda_x^2 - \lambda_y^2)U_{i,0}^0 + \frac{1}{2}\lambda_x^2(U_{i+1,0}^0 + U_{i-1,0}^0) + \frac{1}{2}\lambda_y^2(2U_{i,1}^0)$$

$i = N_x$  のとき

$$U_{N_x,0}^1 = \tau\psi(x_{N_x}, y_0) + (1 - \lambda_x^2 - \lambda_y^2)U_{N_x,0}^0 + \frac{1}{2}\lambda_x^2(2U_{N_x-1,0}^0) + \frac{1}{2}\lambda_y^2(2U_{N_x,1}^0)$$

となる。

また、(2.8) 式に  $j = N_y$  を代入すると

$$U_{i,N_y}^1 = \tau\psi(x_i, y_{N_y}) + (1 - \lambda_x^2 - \lambda_y^2)U_{i,N_y}^0 + \frac{1}{2}\lambda_x^2(U_{i+1,N_y}^0 + U_{i-1,N_y}^0) + \frac{1}{2}\lambda_y^2(2U_{i,N_y-1}^0)$$

となり、ここでも  $i = 0$  で  $U_{-1,N_y}^0$  が  $i = N_x$  で  $U_{N_x+1,N_y}^0$  がでてくるので、 $i$  の値によって場合わけすると

$i = 0$  のとき

$$U_{0,N_y}^1 = \tau\psi(x_0, y_{N_y}) + (1 - \lambda_x^2 - \lambda_y^2)U_{0,N_y}^0 + \frac{1}{2}\lambda_x^2(2U_{1,N_y}^0) + \frac{1}{2}\lambda_y^2(2U_{0,N_y-1}^0)$$

$i = 1, 2, \dots, N_x - 1$  のとき

$$U_{i,N_y}^1 = \tau\psi(x_i, y_{N_y}) + (1 - \lambda_x^2 - \lambda_y^2)U_{i,N_y}^0 + \frac{1}{2}\lambda_x^2(U_{i+1,N_y}^0 + U_{i-1,N_y}^0) + \frac{1}{2}\lambda_y^2(2U_{i,N_y-1}^0)$$

$i = N_x$  のとき

$$U_{N_x,N_y}^1 = \tau\psi(x_{N_x}, y_{N_y}) + (1 - \lambda_x^2 - \lambda_y^2)U_{N_x,N_y}^0 + \frac{1}{2}\lambda_x^2(2U_{N_x-1,N_y}^0) + \frac{1}{2}\lambda_y^2(2U_{N_x,N_y-1}^0)$$

次は  $i = 0$  を式 (2.8) に代入すると、

$$U_{0,j}^1 = \tau\psi(x_0, y_j) + (1 - \lambda_x^2 - \lambda_y^2)U_{0,j}^0 + \frac{1}{2}\lambda_x^2(2U_{1,j}^0) + \frac{1}{2}\lambda_y^2(U_{0,j+1}^0 + U_{0,j-1}^0)$$

となり、ここでは  $j = 0$  で  $U_{0,-1}^0$  が  $j = N_y$  で  $U_{0,N_y+1}^0$  がでてくるので、 $j$  の値によって場合わけすると

$j = 0$  のとき

$$U_{0,0}^1 = \tau\psi(x_0, y_0) + (1 - \lambda_x^2 - \lambda_y^2)U_{0,0}^0 + \frac{1}{2}\lambda_x^2(2U_{1,0}^0) + \frac{1}{2}\lambda_y^2(2U_{0,1}^0)$$

$j = 1, 2, \dots, N_y - 1$  のとき

$$U_{0,j}^1 = \tau\psi(x_0, y_j) + (1 - \lambda_x^2 - \lambda_y^2)U_{0,j}^0 + \frac{1}{2}\lambda_x^2(2U_{1,j}^0) + \frac{1}{2}\lambda_y^2(U_{0,j+1}^0 + U_{0,j-1}^0)$$

$j = N_y$  のとき

$$U_{0,N_y}^1 = \tau\psi(x_0, y_{N_y}) + (1 - \lambda_x^2 - \lambda_y^2)U_{0,N_y}^0 + \frac{1}{2}\lambda_x^2(2U_{1,N_y}^0) + \frac{1}{2}\lambda_y^2(2U_{0,N_y-1}^0)$$



また  $i = N_x$  を式 (2.8) に代入すると、

$$U_{N_x,j}^1 = \tau\psi(x_{N_x}, y_j) + (1 - \lambda_x^2 - \lambda_y^2)U_{N_x,j}^0 + \frac{1}{2}\lambda_x^2(2U_{N_x-1,j}^0) + \frac{1}{2}\lambda_y^2(U_{N_x,j+1}^0 + U_{N_x,j-1}^0)$$

となり、これも  $j$  で場合わけすると

$j = 0$  のとき

$$U_{N_x,0}^1 = \tau\psi(x_{N_x}, y_0) + (1 - \lambda_x^2 - \lambda_y^2)U_{N_x,0}^0 + \frac{1}{2}\lambda_x^2(2U_{N_x-1,0}^0) + \frac{1}{2}\lambda_y^2(2U_{N_x,1}^0)$$

$j = 1, 2, \dots, N_x - 1$  のとき

$$U_{N_x,j}^1 = \tau\psi(x_{N_x}, y_j) + (1 - \lambda_x^2 - \lambda_y^2)U_{N_x,j}^0 + \frac{1}{2}\lambda_x^2(2U_{N_x-1,j}^0) + \frac{1}{2}\lambda_y^2(U_{N_x,j+1}^0 + U_{N_x,j-1}^0)$$

$j = N_y$  のとき

$$U_{N_x,N_y}^1 = \tau\psi(x_{N_x}, y_{N_y}) + (1 - \lambda_x^2 - \lambda_y^2)U_{N_x,N_y}^0 + \frac{1}{2}\lambda_x^2(2U_{N_x-1,N_y}^0) + \frac{1}{2}\lambda_y^2(2U_{N_x,N_y-1}^0)$$

以上の式を見てみると  $U_{0,0}^1, U_{N_x,0}^1, U_{0,N_y}^1, U_{N_x,N_y}^1$  の4つの式がダブっている。これらを一つにまとめて整理すると式 (2.8) は

$$U_{i,0}^1 = \tau\psi(x_i, y_0) + (1 - \lambda_x^2 - \lambda_y^2)U_{i,0}^0 + \frac{1}{2}\lambda_x^2(U_{i+1,0}^0 + U_{i-1,0}^0) + \frac{1}{2}\lambda_y^2(2U_{i,1}^0) \quad (i = 1, 2, \dots, N_x - 1) \quad (2.13)$$

$$U_{i,N_y}^1 = \tau\psi(x_i, y_{N_y}) + (1 - \lambda_x^2 - \lambda_y^2)U_{i,N_y}^0 + \frac{1}{2}\lambda_x^2(U_{i+1,N_y}^0 + U_{i-1,N_y}^0) + \frac{1}{2}\lambda_y^2(2U_{i,N_y-1}^0) \quad (i = 1, 2, \dots, N_x - 1) \quad (2.14)$$

$$U_{0,j}^1 = \tau\psi(x_0, y_j) + (1 - \lambda_x^2 - \lambda_y^2)U_{0,j}^0 + \frac{1}{2}\lambda_x^2(2U_{1,j}^0) + \frac{1}{2}\lambda_y^2(U_{0,j+1}^0 + U_{0,j-1}^0) \quad (j = 1, 2, \dots, N_x - 1) \quad (2.15)$$

$$U_{N_x,j}^1 = \tau\psi(x_{N_x}, y_j) + (1 - \lambda_x^2 - \lambda_y^2)U_{N_x,j}^0 + \frac{1}{2}\lambda_x^2(2U_{N_x-1,j}^0) + \frac{1}{2}\lambda_y^2(U_{N_x,j+1}^0 + U_{N_x,j-1}^0) \quad (j = 1, 2, \dots, N_x - 1) \quad (2.16)$$

$$U_{0,0}^1 = \tau\psi(x_0, y_0) + (1 - \lambda_x^2 - \lambda_y^2)U_{0,0}^0 + \frac{1}{2}\lambda_x^2(2U_{1,0}^0) + \frac{1}{2}\lambda_y^2(2U_{0,1}^0) \quad (i = 0, j = 0) \quad (2.17)$$

$$U_{N_x,0}^1 = \tau\psi(x_{N_x}, y_0) + (1 - \lambda_x^2 - \lambda_y^2)U_{N_x,0}^0 + \frac{1}{2}\lambda_x^2(2U_{N_x-1,0}^0) + \frac{1}{2}\lambda_y^2(2U_{N_x,1}^0) \quad (i = N_x, j = 0) \quad (2.18)$$

$$U_{0,N_y}^1 = \tau\psi(x_0, y_{N_y}) + (1 - \lambda_x^2 - \lambda_y^2)U_{0,N_y}^0 + \frac{1}{2}\lambda_x^2(2U_{1,N_y}^0) + \frac{1}{2}\lambda_y^2(2U_{0,N_y-1}^0) \quad (i = 0, j = N_y) \quad (2.19)$$

$$U_{N_x,N_y}^1 = \tau\psi(x_{N_x}, y_{N_y}) + (1 - \lambda_x^2 - \lambda_y^2)U_{N_x,N_y}^0 + \frac{1}{2}\lambda_x^2(2U_{N_x-1,N_y}^0) + \frac{1}{2}\lambda_y^2(2U_{N_x,N_y-1}^0) \quad (i = N_x, j = N_y) \quad (2.20)$$

となる。

式 (2.6) も同様にして

$$U_{i,0}^{k+1} = 2(1 - \lambda_x^2 - \lambda_y^2)U_{i,0}^k + \lambda_x^2(U_{i+1,0}^k + U_{i-1,0}^k) + \lambda_y^2(2U_{i,1}^k) - U_{i,0}^{k-1}$$

$$(i = 1, 2, \dots, N_x - 1; k \geq 1) \quad (2.21)$$

$$u_{i,N_y}^{k+1} = 2(1 - \lambda_x^2 - \lambda_y^2)U_{i,N_y}^k + \lambda_x^2(U_{i+1,N_y}^k + U_{i-1,N_y}^k) + \lambda_y^2(2U_{i,N_y-1}^k) - U_{i,N_y}^{k-1} \quad (i = 1, 2, \dots, N_x - 1; k \geq 1) \quad (2.22)$$

$$U_{0,j}^{k+1} = 2(1 - \lambda_x^2 - \lambda_y^2)U_{0,j}^k + \lambda_x^2(2U_{1,j}^k) + \lambda_y^2(U_{0,j+1}^k + U_{0,j-1}^k) - U_{0,j}^{k-1} \quad (j = 1, 2, \dots, N_y - 1; k \geq 1) \quad (2.23)$$

$$U_{N_x,j}^{k+1} = 2(1 - \lambda_x^2 - \lambda_y^2)U_{N_x,j}^k + \lambda_x^2(2U_{N_x-1,j}^k) + \lambda_y^2(U_{N_x,j+1}^k + U_{N_x,j-1}^k) - U_{N_x,j}^{k-1} \quad (j = 1, 2, \dots, N_x - 1; k \geq 1) \quad (2.24)$$

$$U_{0,0}^{k+1} = 2(1 - \lambda_x^2 - \lambda_y^2)U_{0,0}^k + \lambda_x^2(2U_{1,0}^k) + \lambda_y^2(2U_{0,1}^k) - U_{0,0}^{k-1} \quad (i = 0, j = 0; k \geq 1) \quad (2.25)$$

$$U_{N_x,0}^{k+1} = 2(1 - \lambda_x^2 - \lambda_y^2)U_{N_x,0}^k + \lambda_x^2(2U_{N_x-1,0}^k) + \lambda_y^2(2U_{N_x,1}^k) - U_{N_x,0}^{k-1} \quad (i = N_x, j = 0; k \geq 1) \quad (2.26)$$

$$U_{0,N_y}^{k+1} = 2(1 - \lambda_x^2 - \lambda_y^2)U_{0,N_y}^k + \lambda_x^2(2U_{1,N_y}^k) + \lambda_y^2(2U_{0,N_y-1}^k) - U_{0,N_y}^{k-1} \quad (i = 0, j = N_y; k \geq 1) \quad (2.27)$$

$$U_{N_x,N_y}^{k+1} = 2(1 - \lambda_x^2 - \lambda_y^2)U_{N_x,N_y}^k + \lambda_x^2(2U_{N_x-1,N_y}^k) + \lambda_y^2(2U_{N_x,N_y-1}^k) - U_{N_x,N_y}^{k-1} \quad (i = N_x, j = N_y; k \geq 1) \quad (2.28)$$

となる。

## 2.3 Wave2d program

```
/** 2次元波動方程式を解くプログラム。
 * 境界条件
 * bc=0    Dirichlet 境界条件
 * bc=1    Neumann 境界条件
 * bc=2    格子点を用いる方法
 *
 * これをコンパイルするには MitsuiWorld が必要です。
 * MitsuiWorld が使える状態であれば普通のコンパイル javac Wave1d.java
 * と普通の起動方法 appletviewer Wave1d.java
 * でOK。
 * ただし appletviewer Wave1d.java としたければコメント欄のどこかに
 *
 * <applet code = Wave2d width=500 height=500></applet>
 *
 * というものが入ってないといけない。
 * それと MitsuiWorld の使い方によっては
 *
 * import Mitsui.*;
 *
 * を消すこと。
 * MitsuiWorld をパッケージとして使わない人はこれを消してください。
 */

import java.applet.*;
import java.awt.*;
import Mitsui.*;      //MitsuiWorld の入っているパッケージ Mitsui を呼び出す。

public class Wave2d extends Applet {
    int nfunc=0;          //これでなんの関数を選んだか判断する。
    int bc=0;             //これでなんの境界条件を選んだか判断する。
    MitsuiWorld_2 m = new MitsuiWorld_2();
    double xmin,xmax,ymin,ymax,zmin,zmax;

    public void init(){
        Graphics g=getGraphics();
        m.setGraphics(g);
        m.setScreenSize(getSize().width,getSize().height);
        //[ a , b ] × [ c , d ] × [ e , f ]
        xmin=-2.0;
        xmax=2.0;
        ymin=-2.0;
        ymax=2.0;
        zmin=-1.0;
        zmax=1.0;
        m.setArea2(xmin,xmax,ymin,ymax,zmin,zmax);
        m.setColor(3);      //グラフの色
    }

    public void paint(Graphics g){

        int nx = 40;        //x の分割数大きすぎると動作が遅い
        int ny = 40;        //y の分割数大きすぎると動作が遅い
        double tau = 0.05;   //時間刻み
        double Tmax = 4.0;   //大きいと描く時間が長くなる
    }
}
```

```

double dx = (xmax-xmin)/nx;
double dy = (ymax-ymin)/ny;
double lambdax = tau / dx;
double lambday = tau / dy;
double lambdax2 = lambdax * lambdax;
double lambday2 = lambday * lambday;

double[] [] u1 = new double[nx+1][ny+1];
double[] [] u2 = new double[nx+1][ny+1];
double[] [] u3 = new double[nx+1][ny+1];

//初期值代入
//phi=
//psi=
for(int i=0;i<=nx;i++)
    for(int j=0;j<=ny;j++)
        u1[i][j] = phi(xmin+i*dx,ymin+j*dy);
for(int i=1;i<nx;i++)
    for(int j=1;j<ny;j++)
        u2[i][j] = (1.0-lambdax2-lambday2) * u1[i][j] +
            0.5 * lambdax2 * (u1[i-1][j]+u1[i+1][j]) +
            0.5 * lambday2 * (u1[i][j-1]+u1[i][j+1]) +
            tau * psi(xmin+i*dx,ymin+j*dy);

if(bc==0){
    //Dirichlet 境界条件
    for(int i=0;i<=nx;i++)
        u2[i][0]=u2[i][ny]=0.0;
    for(int i=0;i<=ny;i++)
        u2[0][i]=u2[nx][i]=0.0;
}
if(bc==1){
    //Neumann 境界条件
    for(int i=0;i<=nx;i++){
        u2[i][0]=u2[i][1];
        u2[i][ny]=u2[i][ny-1];
    }
    for(int i=0;i<=ny;i++){
        u2[0][i]=u2[1][i];
        u2[nx][i]=u2[nx-1][i];
    }
}
if(bc==2){
    //格子点
    for(int i=1;i<nx;i++){
        u2[i][0]=(1.0-lambdax2-lambday2) * u1[i][0] +
            0.5 * lambdax2 * (u1[i-1][0]+u1[i+1][0]) +
            0.5 * lambday2 * (2*u1[i][1]) +
            tau * psi(xmin+i*dx,ymin);
        u2[i][ny]=(1.0-lambdax2-lambday2) * u1[i][ny] +
            0.5 * lambdax2 * (u1[i-1][ny]+u1[i+1][ny]) +
            0.5 * lambday2 * (2*u1[i][ny-1]) +
            tau * psi(xmin+i*dx,ymin+ny*dy);
    }
    for(int j=1;j<ny;j++){
        u2[0][j]=(1.0-lambdax2-lambday2) * u1[0][j] +
            0.5 * lambdax2 * (2*u1[1][j]) +
            0.5 * lambday2 * (u1[0][j-1]+u1[0][j+1]) +
            tau * psi(xmin,ymin+j*dy);
        u2[nx][j]=(1.0-lambdax2-lambday2) * u1[nx][j] +
            0.5 * lambdax2 * (2*u1[nx-1][j]) +

```

```

        0.5 * lambday2 * (u1[nx][j-1]+u1[nx][j+1]) +
        tau * psi(xmin+nx*dx,ymin+j*dy);
    }
    u2[0][0]=(1.0-lambdax2-lambday2) * u1[0][0] +
        0.5 * lambdax2 * (2*u1[1][0]) +
        0.5 * lambday2 * (2*u1[0][1]) +
        tau * psi(xmin,ymin);
    u2[nx][0]=(1.0-lambdax2-lambday2) * u1[nx][0] +
        0.5 * lambdax2 * (2*u1[nx-1][0]) +
        0.5 * lambday2 * (2*u1[nx][1]) +
        tau * psi(xmin+nx*dx,ymin);
    u2[0][ny]=(1.0-lambdax2-lambday2) * u1[0][ny] +
        0.5 * lambdax2 * (2*u1[1][ny]) +
        0.5 * lambday2 * (u1[0][ny-1]) +
        tau * psi(xmin,ymin+ny*dy);
    u2[nx][ny]=(1.0-lambdax2-lambday2) * u1[nx][ny] +
        0.5 * lambdax2 * (2*u1[nx-1][ny]) +
        0.5 * lambday2 * (2*u1[nx][ny-1]) +
        tau * psi(xmin+nx*dx,ymin+ny*dy);

}

//t=0 でのグラフ
m.hideBirdView(u1,nx,ny,1);

//t=0 でのグラフをけす。
g.clearRect(0,30,getSize().width,getSize().height);

//t= でのグラフ
m.hideBirdView(u2,nx,ny,1);

int kmax = (int)Math rint(Tmax/tau);
for(int k=1;k<=kmax;k++){
    for(int i=1;i<nx;i++){
        for(int j=1;j<ny;j++){
            u3[i][j] = 2 * (1.0-lambdax2 -lambday2) * u2[i][j] +
                lambdax2 * (u2[i-1][j]+u2[i+1][j]) +
                lambday2 * (u2[i][j-1]+u2[i][j+1]) - u1[i][j];

            if(bc==0){ //Dirichlet 境界条件
                for(int i=0;i<=nx;i++){
                    u3[i][0] = u3[i][ny] = 0;
                }
                for(int i=0;i<=ny;i++){
                    u3[0][i] = u3[nx][i] = 0;
                }
            }
            if(bc==1){ //Neumann 境界条件
                for(int i=0;i<=nx;i++){
                    u3[i][0] = u3[i][1] ;
                    u3[i][ny]= u3[i][ny-1];
                }
                for(int i=0;i<=ny;i++){
                    u3[0][i] = u3[1][i];
                    u3[nx][i]= u3[nx-1][i];
                }
            }
        }
    }
    if(bc==2){ //仮想格子点
        for(int i=1;i<nx;i++){
            u3[i][0] = 2 * (1.0-lambdax2 -lambday2) * u2[i][0] +

```

```

        lambdax2 * (u2[i-1][0]+u2[i+1][0]) +
        lambday2 * (2*u2[i][1]) - u1[i][0];
    u3[i][ny]= 2 * (1.0-lambdax2 -lambday2) * u2[i][ny] +
        lambdax2 * (u2[i-1][ny]+u2[i+1][ny]) +
        lambday2 * (2*u2[i][ny-1]) - u1[i][ny];
}
for(int j=1;j<ny;j++){
    u3[0][j] = 2 * (1.0-lambdax2 -lambday2) * u2[0][j] +
        lambdax2 * (2*u2[1][j]) +
        lambday2 * (u2[0][j-1]+u2[0][j+1]) - u1[0][j];
    u3[nx][j]= 2 * (1.0-lambdax2 -lambday2) * u2[nx][j] +
        lambdax2 * (2*u2[nx-1][j]) +
        lambday2 * (u2[nx][j-1]+u2[nx][j+1]) - u1[nx][j];
}
u3[0][0] = 2 * (1.0-lambdax2 -lambday2) * u2[0][0] +
    lambdax2 * (2*u2[1][0]) +
    lambday2 * (2*u2[0][1]) - u1[0][0];
u3[nx][0]= 2 * (1.0-lambdax2 -lambday2) * u2[nx][0] +
    lambdax2 * (2*u2[nx-1][0]) +
    lambday2 * (2*u2[nx][1]) - u1[nx][0];
u3[0][ny]= 2 * (1.0-lambdax2 -lambday2) * u2[0][ny] +
    lambdax2 * (2*u2[1][ny]) +
    lambday2 * (2*u2[0][ny-1]) - u1[0][ny];
u3[nx][ny]= 2 * (1.0-lambdax2 -lambday2) * u2[nx][ny] +
    lambdax2 * (2*u2[nx-1][ny]) +
    lambday2 * (2*u2[nx][ny-1]) - u1[nx][ny];
}

//グラフを消す。
g.clearRect(0,30,getSize().width,getSize().height);

//t= k(k>=2) のグラフを書く
m.hideBirdView(u3,nx,ny,1);

//u1 に u2 の値を u2 に u3 の値を代入
for(int i=0;i<=nx;i++){
    for(int j=0;j<=ny;j++){
        u1[i][j] = u2[i][j];
        u2[i][j] = u3[i][j];
    }
}
}
}

public double phi(double x,double y){
    double cx=(xmax+xmin)/2;
    double cy=(ymax+ymin)/2;
    double r =(xmax-xmin)/5;

    if(nfunc==0){
        return Math.sin(Math.PI*x)/2+Math.sin(Math.PI*y)/2;
    }
    if(nfunc==1){
        if((x-cx)*(x-cx)+(y-cy)*(y-cy)<=r*r)
            return Math.sqrt(r*r-(x-cx)*(x-cx)-(y-cy)*(y-cy));
        else
            return 0.0;
    }
}

```

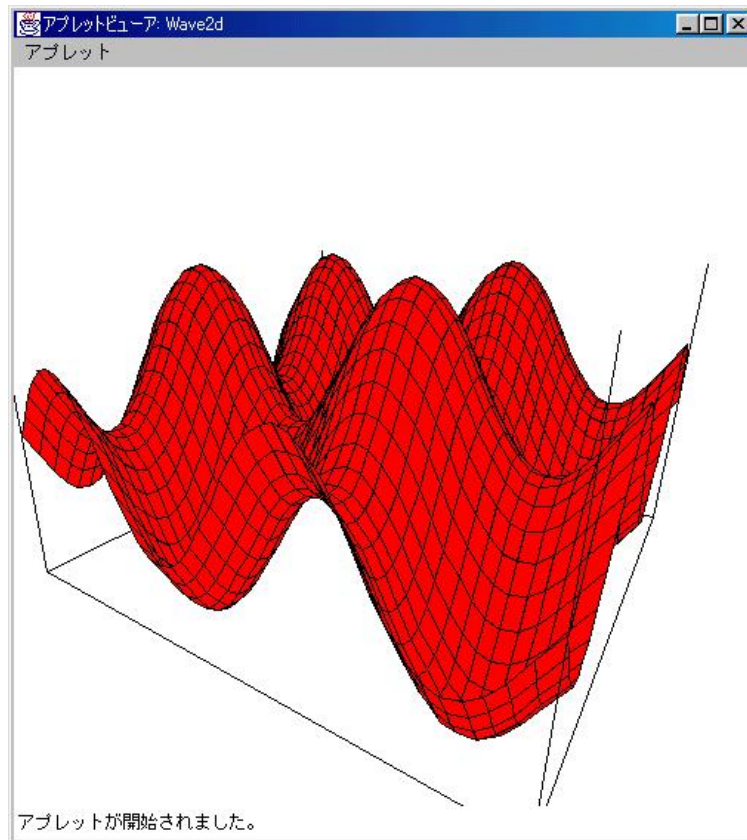
```

    if(nfunc==2){
        if(xmin<=x && x<=xmin+1)
            return (Math.sin(Math.PI*(x*2-0.5))+1)/4;
        else
            return 0.0;
    }
    if(nfunc == 3){
        if(xmin<=x && x<=xmin+1)
            return (Math.sin(Math.PI*(x*2-0.5))+1)/4;
        if(xmax-1<=x && x<=xmax)
            return (Math.sin(Math.PI*((x-xmax+1)*2-0.5))+1)/6;
        else
            return 0.0;
    }
    if(nfunc==4){
        if(x*x+y*y<1)
            return ((Math.cos(Math.PI*x)+Math.cos(Math.PI*y))+1)/2;
        else
            return 0.0;
    }
    else
        return 0.0;
}

//初期条件
public double psi(double x,double y){
    double cx=(xmax+xmin)/2;
    double cy=(ymax+ymin)/2;
    double r =(xmax-xmin)/5;

    if(nfunc==2){
        if(xmin<=x && x<=xmin+1)
            return -2*Math.PI*Math.cos(Math.PI*(x*2-0.5))/4;
        else
            return 0.0;
    }
    if(nfunc == 3){
        if(xmin<=x && x<=xmin+1)
            return -2*Math.PI*Math.cos(Math.PI*(x*2-0.5))/4;
        if(xmax-1<=x && x<=xmax)
            return 2*Math.PI*Math.cos(Math.PI*((x-xmax+1)*2-0.5))/6;
        else
            return 0.0;
    }
    else
        return 0.0;
}
}

```





## 第3章 Javaによるアニメーション

Javaにはアプレットという機能がついており、これはWebブラウザ上で実行できることからインターネット上での公開が非常に楽に行えて、さらにGUIを標準で装備しているのでテキストの入力やボタンの設置等を簡単に行うことが出来る。またマルチスレッドプログラミングという機能も標準で装備されておりアニメーションを楽に作ることができる。今回の僕の卒論の最大のテーマのでもある「世界中の人々に手軽に数値解析を体感してもらう」。このテーマにそったプログラムを作るにあたって上のそれらの機能をフルに使う。アプレット機能でWeb上で公開できるプログラムにし、GUIを作り操作性を良くし世界中の人々に気軽に扱えるプログラムにし、マルチスレッド機能を駆使してGUIを有効にし、アニメーションを思い通りに動かすプログラミングを作る。この章ではそのプログラムの紹介とそのプログラムの解説をする。

### 3.1 WaveAll program

```
/** 1次元波動方程式と2次元波動方程式を求めるプログラミングです。
 * マルチスレッドプログラミングを使っています。
 * ダブルバッファリングを使っています。
 *
 * これをコンパイルするには MitsuiWorld が必要です。
 * MitsuiWorld が使える状態であれば普通のコンパイル javac WaveAll_2_0.java
 * と普通の起動方法 appletviewer Wave1d.java
 * でOK。
 * ただし appletviewer WaveAll_2_0.java としたければコメント欄のどこかに
 *
 * <applet code = WaveAll_2_0 width=500 height=500></applet>
 *
 * というものが入ってないといけない。
 * それと MitsuiWorld の使い方によっては
 *
 * import Mitsui.*;
 *
 * を消すこと。
 * MitsuiWorld をパッケージとして使わない人はこれを消してください
 *
 */

import java.applet.*;
import java.awt.*;
import Mitsui.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
```

```

import java.util.*;

public class WaveAll_2_0 extends Applet
    implements Runnable, ActionListener, ItemListener{

    Thread th = null;                                //スレッド
    Button b_start, b_stop, b_next, b_back, b_reverse;    //各ボタン
    Button b_1dset, b_2dset;
    Label l_time, l_area;                                //各ラベル
    Label2 l_func;
    Choice c_dimension, c_boundary;                    //関数を選ぶ GUI
    boolean runmove=false;                             //スレッドが動いてるか止まっているか
    boolean reverseflag=false;                         //reverse の時に時間をマイナスにする

    int im_t, re_t;                                    //時間の経過
    int wmax, hmax;                                    //画面のサイズ

    //もう一つの画面の定義
    Graphics bg;
    Image buf;

    //画面作成に必要なもの
    MitsuiWorld_2 m;
    // 2次元
    double xmin2, xmax2, ymin2, ymax2, zmin2, zmax2;
    int nx2, ny2;
    double dx2, dy2;
    double tau2;
    // 1次元
    double xmin1, xmax1, ymin1, ymax1;
    int nx1;
    double dx1;
    double tau1;

    int n1=100;
    int n2=50;

    double[][] u2_1 = new double[n2+1][n2+1];
    double[][] u2_2 = new double[n2+1][n2+1];
    double[][] u2_3 = new double[n2+1][n2+1];
    double[] u1_1 = new double[n1+1];
    double[] u1_2 = new double[n1+1];
    double[] u1_3 = new double[n1+1];

    int nfunc1=0;                                     // 1次元の関数を変える
    int nfunc2=0;                                     // 2次元の関数を変える
    int dimension=0;                                  //次元を変える/0= 1次元, 1= 2次元
    int boundary=0;                                   //境界条件を変える

    //視覚を変える変数
    int lx, ly, lz;                                   //各軸の長さ
    int angx, angy;                                   //視覚の角度
    int x0, y0;                                       //原点の位置

    //初期設定
    public void init(){
        setLayout(new BorderLayout());
        Panel p = new Panel();                        //NORTH 地区を枠決めしてる
    }

```

```

p.setLayout(new GridLayout(3,1,0,0));
Panel p1 = new Panel();           // 1 段目
Panel p2 = new Panel();           // 2 段目
Panel p3 = new Panel();           // 3 段目

//start ボタンの設置
b_start=new Button("START");
b_start.addActionListener(this);
p2.add(b_start);

//stop ボタンの設置
b_stop=new Button("STOP");
b_stop.addActionListener(this);
p2.add(b_stop);

//reverse ボタンの設置
b_reverse=new Button("REVERSE");
b_reverse.addActionListener(this);
p2.add(b_reverse);

//back ボタンの設置
b_back=new Button("BACK");
b_back.addActionListener(this);
p2.add(b_back);

//next ボタンの設置
b_next=new Button("NEXT");
b_next.addActionListener(this);
p2.add(b_next);

//dimension チョイスの設置
c_dimension=new Choice();
c_dimension.addItem(" 1 次元波動方程式");
c_dimension.addItem(" 2 次元波動方程式");
c_dimension.addItemListener(this);
p1.add(c_dimension);

//boundary チョイスの設置
c_boundary=new Choice();
c_boundary.addItem("Dirichlet");
c_boundary.addItem("Neumann");
c_boundary.addItem("仮想格子点");
c_boundary.addItemListener(this);
p1.add(c_boundary);

// 1 次元のセッティングを行うボタン
b_1dset = new Button("1Dsetting");
b_1dset.addActionListener(this);
p1.add(b_1dset);

// 2 次元のセッティングを行うボタン
b_2dset = new Button("2Dsetting");
b_2dset.addActionListener(this);
p1.add(b_2dset);

//ラベル l_time の設置 時間を表示する。
l_time = new Label(" ");

```

```

p2.add(l_time);

//ラベル l_area の設置 描写範囲の表示
l_area = new Label(" ");
p3.add(l_area);

//ラベル l_func の設置 選んだ関数の表示
l_func = new Label2(" =sin( x) ",
                    " =0");
p3.add(l_func);

p.add(p1);
p.add(p2);
p.add(p3);
add(p,"North");

//画面の横と縦の長さを得る。
wmax = getSize().width;
hmax = getSize().height;

//描写範囲を決める
// 2次元 [xmin2,xmax2] × [ymin2,ymax2] × [zmin2,zmax2]
xmin2 = -2.0;
xmax2 = 2.5;
ymin2 = -2.0;
ymax2 = 2.5;
zmin2 = -1.0;
zmax2 = 1.0;

// 1次元 [xmin1,xmax1] × [ymin1,ymax1]
xmin1 = -0.2;
xmax1 = 1.2;
ymin1 = -1.2;
ymax1 = 1.8;

//作図用の描写画面の作成
buf = createImage(wmax,hmax);
bg = buf.getGraphics();

//MitsuiWorld
m = new MitsuiWorld_2();
m.setGraphics(bg);
m.setScreenSize(wmax,hmax);

//分割数
nx1 = 100; //1次元のx分割数
nx2 = 40; //2次元のxの分割数大きすぎると動作が遅い
ny2 = 40; //2次元のyの分割数大きすぎると動作が遅い

//時刻間隔
tau1 = 0.01;
tau2 = 0.07;

//視覚を変える変数の初期値
lx=0;
ly=0;
lz=0;
x0=y0=0;

```

```

    angx=angy=0;
}

//起動と同時にスレッドを走らせる
public void start(){
    if(th==null){
        th=new Thread(this);
        th.start();
    }
}

//スレッドの実装
public void run(){
    while(true){
        while(!runmove){
            //起動中スレッドを走らせる。
            //start ボタンを押すことによって
            //ここからぬけ、スレッドが有効になる。
            if(!reverseflag)
                re_t++;
            else
                re_t--;

            im_t++;

            bg.clearRect(0,0,wmax,hmax);
            if(dimension==0){
                //画面をクリア
                // 1次元波動方程式のとき
                l_time.setText("t="+re_t*tau1);
                double[] u = f1(im_t);
                m.setColor(Color.black);
                drawAxis();
                m.setColor(Color.pink);
                m.move(0.0,u[0]);
                for(int i=1;i<=nx1;i++)
                    m.draw(i*dx1,u[i]);
            }
            if(dimension==1){
                // 2次元波動方程式のとき
                l_time.setText("t="+re_t*tau2);
                double[][] u = f2(im_t);
                //計算した結果を格納する
                //メモリ内で描く
                m.hideBirdView(u,nx2,ny2,1);
            }
            repaint();
            try{
                Thread.sleep(100);
            }catch(InterruptedException e){}
        }
    }
}

//終了と同時にスレッドも終了
public void stop(){
    if(th!=null){
        th = null;
    }
}

public void actionPerformed(ActionEvent e){
    //start ボタンを押したときのアクション
    if(e.getSource()==b_start){
        runmove=true;
        //再びグラフを描き始める
    }
}

```

```

//stop ボタンを押したときのアクション
if(e.getSource()==b_stop){
    if(!runmove){
        reverseflag=false;
        im_t=0;
        re_t=0;
        repaint();
    }
    else{
        runmove=false;
    }
}

//back ボタンを押したときのアクション
if(e.getSource()==b_back){
    if(!reverseflag)
        re_t--;
    else
        re_t++;

    if(!runmove){
        if(dimension==0){
            for(int i=0;i<=nx1;i++){
                u1_2[i] = u1_1[i];
                u1_1[i] = u1_3[i];
            }
            double[] u = f1(im_t);
            for(int i=0;i<=nx1;i++){
                u1_2[i] = u1_1[i];
                u1_1[i] = u1_3[i];
                u1_3[i] = u1_2[i];
            }

            l_time.setText("t="+((double)re_t*tau1));
            bg.clearRect(0,0,wmax,hmax);
            m.setColor(Color.black);
            drawAxis();
            m.setColor(Color.pink);
            m.move(0.0,u1_3[0]);
            for(int i=1;i<=nx1;i++)
                m.draw(i*dx1,u1_3[i]);
        }
        if(dimension==1){
            for(int i=0;i<=nx2;i++){
                for(int j=0;j<=ny2;j++){
                    u2_2[i][j] = u2_1[i][j];
                    u2_1[i][j] = u2_3[i][j];
                }
            }
            double[][] u = f2(im_t);
            for(int i=0;i<=nx2;i++){
                for(int j=0;j<=ny2;j++){
                    u2_2[i][j] = u2_1[i][j];
                    u2_1[i][j] = u2_3[i][j];
                    u2_3[i][j] = u2_2[i][j];
                }
            }
        }
    }
}

```

```

        l_time.setText("t="+double)re_t*tau2);
        bg.clearRect(0,0,wmax,hmax);          //画面をクリア
        m.hideBirdView(u2_3,nx2,ny2,1);        //メモリ内で描く
    }
    repaint();
}

//next ボタンを押したときのアクション
if(e.getSource()==b_next){
    if(!runmove){                                //一時停止状態のときのみ
        if(!reverseflag)
            re_t++;
        else
            re_t--;                                //次の時間のグラフを見れる。

        bg.clearRect(0,0,wmax,hmax);            //画面をクリア
        if(dimension==0){
            l_time.setText("t="+double)re_t*tau1);
            double[] u = f1(im_t);
            m.setColor(Color.black);
            drawAxis();
            m.setColor(Color.pink);
            m.move(0.0,u[0]);
            for(int i=1;i<=nx1;i++){
                m.draw(i*dx1,u[i]);
            }
            if(dimension==1){
                l_time.setText("t="+double)re_t*tau2);
                double[][] u = f2(im_t);          //計算した結果を格納する
                m.hideBirdView(u,nx2,ny2,1);      //メモリ内で描く
            }
            repaint();
        }
    }

//reverse ボタンを押したときのアクション
if(e.getSource()==b_reverse){
    //逆流してるかしてないか。
    if(reverseflag)
        reverseflag=false;
    if(!reverseflag)
        reverseflag=true;

    if(dimension==0){
        double[]u = f1(im_t);
        for(int i=0;i<=nx1;i++){
            u1_2[i] = u1_1[i];
            u1_1[i] = u1_3[i];
            u1_3[i] = u1_2[i];
        }
    }
    if(dimension==1){
        double[][] u = f2(im_t);
        for(int i=0;i<=nx2;i++){
            for(int j=0;j<=ny2;j++){
                u2_2[i][j]=u2_1[i][j];
            }
        }
    }
}

```

```

        u2_1[i][j]=u2_3[i][j];
        u2_3[i][j]=u2_2[i][j];
    }
}
}

// 1次元波動方程式の設定をするダイアログを作っている。
if(e.getSource()==b_1dset){
    String s = ("初期条件を選んでください");

    //初期値選択欄の作成
    Choice c_func1 = new Choice();
    c_func1.addItem("ひたすら振動を繰り返す波");
    c_func1.addItem("別れては重なる波");
    c_func1.addItem("右方向に流れていく波");
    c_func1.addItem("二つの波の合体");

    //分割数記入欄の作成
    NumericField inputN = new NumericField(""+nx1);
    inputN.setBorder
        (new TitledBorder("分割数を指定してください(<=100)"));

    //時刻記入欄の作成
    NumericField inputTau = new NumericField(""+tau1);
    inputTau.setBorder
        (new TitledBorder("時刻を指定してください"));

    //描写範囲指定欄の作成
    NumericField inputArea =
        new NumericField(xmin1+" "+xmax1+" "+ymin1+" "+ymax1);
    inputArea.setBorder
        (new TitledBorder("描写範囲を指定してください 数の間には空白を入れて下さい。"));

    //オブジェクトを一つにまとめて
    Object[] obj = {s,c_func1,inputN,inputTau,inputArea};

    //ダイアログの作成
    int ans = JOptionPane.showConfirmDialog(this,obj,
        "1次元波動方程式の設定",
        JOptionPane.OK_CANCEL_OPTION);

    if(ans==0){ //OKを押した時
        nfunc1 = c_func1.getSelectedIndex();

        if(dimension==0){
            //式の記述
            switch(nfunc1){
                case 0:
                    l_func.setText(" =sin( x)"," =0");
                    break;
                case 1:
                    l_func.setText(" =8x-3(3/8<x<=1/2),-8x+5(1/2<x<=5/8)",
                        " =0");
                    break;
                case 2:
                    l_func.setText(" =(sin( (10x-0.5))+1)/4",
                        " =-2.5 cos( (10x-0.5))");
            }
        }
    }
}

```



```

        break;
    case 3:
        l_func.setText(" 1=(sin( (10x-0.5))+1)/4",
            " 2=(sin( (20(x-0.9)-0.5))+1)/8");
        break;
    default:
        l_func.setText("未定","未定");
        break;
    }
}

nx1 = Integer.parseInt(inputN.getText().trim());
tau1 = Double.valueOf(inputTau.getText().trim()).doubleValue();

//描写範囲を読み込む時は「分割」をつかう。
String str = inputArea.getText().trim();
//空白区切りで分割する
StringTokenizer st = new StringTokenizer(str," ");
if(st.countTokens()==4){ //分割数が4のとき
    xmin1 = Double.valueOf(st.nextToken()).doubleValue();
    xmax1 = Double.valueOf(st.nextToken()).doubleValue();
    ymin1 = Double.valueOf(st.nextToken()).doubleValue();
    ymax1 = Double.valueOf(st.nextToken()).doubleValue();
}

runmove = false; //初期状態に戻す
reverseflag=false;
im_t=0;
re_t=0;
repaint();
}
}

if(e.getSource()==b_2dset){
    String s = ("初期条件を選んでください");

    Choice c_func2 = new Choice();
    c_func2.addItem(" =sin( x)+sin( y), =0");
    c_func2.addItem("定常波");
    c_func2.addItem("平面波");
    c_func2.addItem("平面波の合体");
    c_func2.addItem("変な波");
    c_func2.addItem("変な波右方向");

    //ダイアログは持ち込むオブジェクトを縦に並べる。しかし
    //パネルを使うことにより横にもコンテンツを増やした。
    Panel pdivide = new Panel();
    JLabel N = new JLabel("分割数");
    NumericField inputNx = new NumericField(""+nx2,10);
    NumericField inputNy = new NumericField(""+ny2,10);
    inputNx.setBorder(new TitledBorder("Nx(<=50)"));
    inputNy.setBorder(new TitledBorder("Ny(<=50)"));
    pdivide.add(N);
    pdivide.add(inputNx);
    pdivide.add(inputNy);

    NumericField inputTau = new NumericField(""+tau2);

```

```

inputTau.setBorder
    (new TitledBorder("時刻刻みを指定してください"));

NumericField inputArea =
    new NumericField(xmin2+" "+xmax2+"          "+
                    ymin2+" "+ymax2+"          "+zmin2+" "+zmax2);
inputArea.setBorder
    (new TitledBorder("描写範囲の指定 数の間には空白を入れて下さい。"));
//角度を変える欄の作成
Panel pangle = new Panel();
JLabel A = new JLabel("角度変え (ラジアン)");
NumericField inputAngx = new NumericField(""+angx,10);
NumericField inputAngy = new NumericField(""+angy,10);
inputAngx.setBorder(new TitledBorder("水平方向"));
inputAngy.setBorder(new TitledBorder("垂直方向"));
pangle.add(A);
pangle.add(inputAngx);
pangle.add(inputAngy);

//原点移動の欄の作成
Panel porigin = new Panel();
JLabel O = new JLabel("原点移動 (ピクセル)");
NumericField inputX0 = new NumericField(""+x0,10);
NumericField inputY0 = new NumericField(""+y0,10);
inputX0.setBorder(new TitledBorder("右向き"));
inputY0.setBorder(new TitledBorder("上向き"));
porigin.add(O);
porigin.add(inputX0);
porigin.add(inputY0);

//軸の長さ変更欄の作成
Panel plength = new Panel();
JLabel L = new JLabel("サイズ変え");
NumericField inputLx = new NumericField(""+lx,8);
NumericField inputLy = new NumericField(""+ly,8);
NumericField inputLz = new NumericField(""+lz,8);
inputLx.setBorder(new TitledBorder("x 軸"));
inputLy.setBorder(new TitledBorder("y 軸"));
inputLz.setBorder(new TitledBorder("z 軸"));
plength.add(L);
plength.add(inputLx);
plength.add(inputLy);
plength.add(inputLz);

Object[] obj = {s,c_func2,pdivide,inputTau,inputArea,pangle,
                porigin,plength};

int ans = JOptionPane.showConfirmDialog(this,obj,
                                         "2次元波動方程式の設定",
                                         JOptionPane.OK_CANCEL_OPTION);

if(ans==0){
    nfunc2 = c_func2.getSelectedIndex();

    if(dimension==1){
        //式の記述
        switch(nfunc2){
            case 0:
                l_func.setText(" =sin( x)+sin( y)",

```

```

        " =0");
        break;
    case 1:
        l_func.setText("m=(A+B)/2,n=(C+D)/2,r=(B-A)/5",
            " =sqrt(r^2-(x-m)^2-(x-n)^2)(m^2+n^2<=r^2)");
        break;
    case 2:
        l_func.setText(" =(sin( (2x-0.5))+1)/4",
            " =-2 cos( (2x-0.5))/4");
        break;
    case 3:
        l_func.setText(" 1=(sin( (2x-0.5))+1)/4",
            " 2=(sin( (2(x-B+1)-0.5))+1)/6");
        break;
    default:
        l_func.setText("未定","未定");
        break;
    }
}

nx2 = Integer.parseInt(inputNx.getText().trim());
ny2 = Integer.parseInt(inputNy.getText().trim());

tau2 = Double.valueOf(inputTau.getText().trim()).doubleValue();

String str = inputArea.getText().trim();
StringTokenizer st = new StringTokenizer(str, " ");
if(st.countTokens()==6){
    xmin2 = Double.valueOf(st.nextToken()).doubleValue();
    xmax2 = Double.valueOf(st.nextToken()).doubleValue();
    ymin2 = Double.valueOf(st.nextToken()).doubleValue();
    ymax2 = Double.valueOf(st.nextToken()).doubleValue();
    zmin2 = Double.valueOf(st.nextToken()).doubleValue();
    zmax2 = Double.valueOf(st.nextToken()).doubleValue();
}

angx = Integer.parseInt(inputAngx.getText().trim());
angy = Integer.parseInt(inputAngy.getText().trim());

x0 = Integer.parseInt(inputX0.getText().trim());
y0 = Integer.parseInt(inputY0.getText().trim());

lx = Integer.parseInt(inputLx.getText().trim());
ly = Integer.parseInt(inputLy.getText().trim());
lz = Integer.parseInt(inputLz.getText().trim());

runmove = false; //初期状態に戻す
reverseflag=false;
im_t=0;
re_t=0;
repaint();
}
}

//dimension,boundary,fund チョイスを選んだとき
public void itemStateChanged(ItemEvent e){
    dimension=c_dimension.getSelectedIndex(); //次元番号を変える

```

```

        boundary = c_boundary.getSelectedIndex(); //境界条件番号を変える
        runmove = false; //初期状態に戻す
        reverseflag=false;
        im_t=0;
        re_t=0;
        repaint();
    }

    public void paint(Graphics g){
        if(im_t==0){
            bg.clearRect(0,0,wmax,hmax);
            if(dimension==0){
                dx1 = 1.0/nx1; // 1次元のx格子点
                l_time.setText("t=0.0"); //時刻と描写範囲の表示
                l_area.setText("[ "+xmin1+" , "+xmax1+" ] × [ "+ymin1+" , "+ymax1+" ]");
                m.setArea(xmin1,xmax1,ymin1,ymax1);

                double[] u = f1(im_t);
                m.setColor(Color.black);
                drawAxis();
                m.setColor(Color.pink);
                m.move(0.0,u[0]);
                for(int i=1;i<=nx1;i++){
                    m.draw(i*dx1,u[i]);
                }
            }
            if(dimension==1){
                dx2 = (xmax2-xmin2)/nx2; // 2次元のx格子点
                dy2 = (ymax2-ymin2)/ny2; // 2次元のy格子点
                l_time.setText("t=0.0"); //時刻と描写範囲の表示
                l_area.setText("[ "+xmin2+" , "+xmax2+" ] × [ "+ymin2+" , "+ymax2+"
                    " ] × [ "+zmin2+" , "+zmax2+" ]");
                //視覚を変えている
                m.changeAngle(angx,angy);
                m.changePosition(x0,y0);
                m.changeSize(lx,ly,lz);

                //描き始める。
                m.setArea2(xmin2,xmax2,ymin2,ymax2,zmin2,zmax2);
                double[][] u = f2(im_t);
                m.setColor(Color.pink);
                m.hideBirdView(u,nx2,ny2,1);
            }
        }
        g.drawImage(buf,0,0,this); //もう一つの画面をくっつける
    }

    public double[][] f2(int t){
        double lambdax = tau2 / dx2;
        double lambday = tau2 / dy2;
        double lambdax2 = lambdax * lambdax;
        double lambday2 = lambday * lambday;

        if(t==0){ //t=0 のとき
            for(int i=0;i<=nx2;i++){
                for(int j=0;j<=ny2;j++){
                    u2_1[i][j] = phi(xmin2+i*dx2,ymin2+j*dy2); //初期値代入
                }
            }
        }

        return u2_1;
    }

```

```

}

if(t==1){
    for(int i=1;i<nx2;i++){
        for(int j=1;j<ny2;j++){
            u2_2[i][j] = (1.0-lambdax2-lambday2) * u2_1[i][j] +
                0.5 * lambdax2 * (u2_1[i-1][j]+u2_1[i+1][j]) +
                0.5 * lambday2 * (u2_1[i][j-1]+u2_1[i][j+1]) +
                tau2 * psi(xmin2+i*dx2,ymin2+j*dy2);

            if(boundary==0){ //Dirichlet 境界条件
                for(int i=0;i<=nx2;i++){
                    u2_2[i][0]=u2_2[i][ny2]=0.0;
                }
                for(int i=0;i<=ny2;i++){
                    u2_2[0][i]=u2_2[nx2][i]=0.0;
                }
            }
            if(boundary==1){ //Neumann 境界条件
                for(int i=0;i<=nx2;i++){
                    u2_2[i][0]=u2_2[i][1];
                    u2_2[i][ny2]=u2_2[i][ny2-1];
                }
                for(int i=0;i<=ny2;i++){
                    u2_2[0][i]=u2_2[1][i];
                    u2_2[nx2][i]=u2_2[nx2-1][i];
                }
            }
        }
    }
    if(boundary==2){ //仮想格子点
        for(int i=1;i<nx2;i++){
            u2_2[i][0]=(1.0-lambdax2-lambday2) * u2_1[i][0] +
                0.5 * lambdax2 * (u2_1[i-1][0]+u2_1[i+1][0]) +
                0.5 * lambday2 * (2*u2_1[i][1]) +
                tau2 * psi(xmin2+i*dx2,ymin2);
            u2_2[i][ny2]=(1.0-lambdax2-lambday2) * u2_1[i][ny2] +
                0.5 * lambdax2 * (u2_1[i-1][ny2]+u2_1[i+1][ny2]) +
                0.5 * lambday2 * (2*u2_1[i][ny2-1]) +
                tau2 * psi(xmin2+i*dx2,ymin2+ny2*dy2);
        }
        for(int j=1;j<ny2;j++){
            u2_2[0][j]=(1.0-lambdax2-lambday2) * u2_1[0][j] +
                0.5 * lambdax2 * (2*u2_1[1][j]) +
                0.5 * lambday2 * (u2_1[0][j-1]+u2_1[0][j+1]) +
                tau2 * psi(xmin2,ymin2+j*dy2);
            u2_2[nx2][j]=(1.0-lambdax2-lambday2) * u2_1[nx2][j] +
                0.5 * lambdax2 * (2*u2_1[nx2-1][j]) +
                0.5 * lambday2 * (u2_1[nx2][j-1]+u2_1[nx2][j+1]) +
                tau2 * psi(xmin2+nx2*dx2,ymin2+j*dy2);
        }
        u2_2[0][0]=(1.0-lambdax2-lambday2) * u2_1[0][0] +
            0.5 * lambdax2 * (2*u2_1[1][0]) +
            0.5 * lambday2 * (2*u2_1[0][1]) +
            tau2 * psi(xmin2,ymin2);
        u2_2[nx2][0]=(1.0-lambdax2-lambday2) * u2_1[nx2][0] +
            0.5 * lambdax2 * (2*u2_1[nx2-1][0]) +
            0.5 * lambday2 * (2*u2_1[nx2][1]) +
            tau2 * psi(xmin2+nx2*dx2,ymin2);
        u2_2[0][ny2]=(1.0-lambdax2-lambday2) * u2_1[0][ny2] +
            0.5 * lambdax2 * (2*u2_1[1][ny2]) +
            0.5 * lambday2 * (u2_1[0][ny2-1]) +

```

```

        tau2 * psi(xmin2,ymin2+ny2*dy2);
u2_2[nx2][ny2]=(1.0-lambdax2-lambday2) * u2_1[nx2][ny2] +
0.5 * lambdax2 * (2*u2_1[nx2-1][ny2]) +
0.5 * lambday2 * (2*u2_1[nx2][ny2-1]) +
tau2 * psi(xmin2+nx2*dx2,ymin2+ny2*dy2);

    }
    return u2_2;
}

else{
//t>=2 のとき
    for(int i=1;i<nx2;i++)
        for(int j=1;j<ny2;j++)
            u2_3[i][j] = 2 * (1.0-lambdax2 -lambday2) * u2_2[i][j] +
            lambdax2 * (u2_2[i-1][j]+u2_2[i+1][j]) +
            lambday2 * (u2_2[i][j-1]+u2_2[i][j+1]) - u2_1[i][j];

    if(boundary==0){
//Dirichlet 境界条件
        for(int i=0;i<nx2;i++)
            u2_3[i][0]=u2_3[i][ny2]=0.0;
        for(int i=0;i<ny2;i++)
            u2_3[0][i]=u2_3[nx2][i]=0.0;
    }
    if(boundary==1){
//Neumann 境界条件
        for(int i=0;i<nx2;i++){
            u2_3[i][0] = u2_3[i][1] ;
            u2_3[i][ny2]= u2_3[i][ny2-1];
        }
        for(int i=0;i<ny2;i++){
            u2_3[0][i] = u2_3[1][i];
            u2_3[nx2][i]= u2_3[nx2-1][i];
        }
    }
    if(boundary==2){
//仮想格子点
        for(int i=1;i<nx2;i++){
            u2_3[i][0] = 2 * (1.0-lambdax2 -lambday2) * u2_2[i][0] +
            lambdax2 * (u2_2[i-1][0]+u2_2[i+1][0]) +
            lambday2 * (2*u2_2[i][1]) - u2_1[i][0];
            u2_3[i][ny2]=2 * (1.0-lambdax2 -lambday2) * u2_2[i][ny2] +
            lambdax2 * (u2_2[i-1][ny2]+u2_2[i+1][ny2]) +
            lambday2 * (2*u2_2[i][ny2-1]) - u2_1[i][ny2];
        }
        for(int j=1;j<ny2;j++){
            u2_3[0][j] = 2 * (1.0-lambdax2 -lambday2) * u2_2[0][j] +
            lambdax2 * (2*u2_2[1][j]) +
            lambday2 * (u2_2[0][j-1]+u2_2[0][j+1]) - u2_1[0][j];
            u2_3[nx2][j]=2 * (1.0-lambdax2 -lambday2) * u2_2[nx2][j] +
            lambdax2 * (2*u2_2[nx2-1][j]) +
            lambday2 * (u2_2[nx2][j-1]+u2_2[nx2][j+1])-u2_1[nx2][j];
        }
        u2_3[0][0] = 2 * (1.0-lambdax2 -lambday2) * u2_2[0][0] +
            lambdax2 * (2*u2_2[1][0]) +
            lambday2 * (2*u2_2[0][1]) - u2_1[0][0];
        u2_3[nx2][0]= 2 * (1.0-lambdax2 -lambday2) * u2_2[nx2][0] +
            lambdax2 * (2*u2_2[nx2-1][0]) +
            lambday2 * (2*u2_2[nx2][1]) - u2_1[nx2][0];
        u2_3[0][ny2]= 2 * (1.0-lambdax2 -lambday2) * u2_2[0][ny2] +
            lambdax2 * (2*u2_2[1][ny2]) +

```

```

        lambday2 * (2*u2_2[0][ny2-1]) - u2_1[0][ny2];
u2_3[nx2][ny2]=2 * (1.0-lambdax2 -lambday2) * u2_2[nx2][ny2] +
        lambdax2 * (2*u2_2[nx2-1][ny2]) +
        lambday2 * (2*u2_2[nx2][ny2-1]) - u2_1[nx2][ny2];
    }

    //u1 に u2 の値を u2 に u3 の値を代入
    for(int i=0;i<=nx2;i++){
        for(int j=0;j<=ny2;j++){
            u2_1[i][j] = u2_2[i][j];
            u2_2[i][j] = u2_3[i][j];
        }
    }

    return u2_3;
}

}

public double[] f1(int t){
    double lambda = tau1/dx1;
    double lambda2 = lambda * lambda;

    if(t==0){
        for(int i=0;i<=nx1;i++){
            u1_1[i] = phi(i*dx1);
        }
        return u1_1;
    }

    if(t==1){
        for(int i=1;i<nx1;i++){
            u1_2[i] = (1-lambda2) * u1_1[i] +
                0.5 * lambda2 * (u1_1[i-1]+u1_1[i+1]) + tau1 * psi(i*dx1);

            if(boundary==0){ //Dirichlet 境界条件
                u1_2[0]=u1_2[nx1]=0;
            }
            if(boundary==1){ //Neumann 境界条件
                u1_2[0]=u1_2[1];
                u1_2[nx1]=u1_2[nx1-1];
            }
            if(boundary==2){ //仮想格子点
                u1_2[0] = (1-lambda2) * u1_1[0] +
                    0.5 * lambda2 * (2*u1_1[1]) + tau1 * psi(0.0);
                u1_2[nx1] = (1-lambda2) * u1_1[nx1] +
                    0.5 * lambda2 * (2*u1_1[nx1-1]) + tau1 * psi(nx1*dx1);
            }

            return u1_2;
        }
    }

    else{
        for(int i=1;i<nx1;i++){
            u1_3[i] = 2 * (1.0-lambda2) * u1_2[i] +
                lambda2 * (u1_2[i-1]+u1_2[i+1]) - u1_1[i];

            if(boundary==0){ //Dirichlet 境界条件
                u1_3[0] = u1_3[nx1] = 0;
            }
        }
    }
}

```

```

    }
    if(boundary==1){
        //Neumann 境界条件
        u1_3[0]=u1_3[1];
        u1_3[nx1]=u1_3[nx1-1];
    }
    if(boundary==2){
        //仮想格子点
        u1_3[0] = 2 * (1.0-lambda2) * u1_2[0] +
            lambda2 * (2*u1_2[1]) - u1_1[0];
        u1_3[nx1] = 2 * (1.0-lambda2) * u1_2[nx1] +
            lambda2 * (2*u1_2[nx1-1]) - u1_1[nx1];
    }

    //u1 に u2 の値を u2 に u3 の値を代入
    for(int i=0;i<nx1;i++){
        u1_1[i] = u1_2[i];
        u1_2[i] = u1_3[i];
    }

    return u1_3;
}
}

//初期値
public double phi(double x,double y){
    double cx = (xmax2+xmin2)/2;
    double cy = (ymax2+ymin2)/2;
    double r = (xmax2-xmin2)/5;

    if(nfunc2==0){
        return Math.sin(Math.PI*x)/2+Math.sin(Math.PI*y)/2;
    }
    if(nfunc2==1){
        if((x-cx)*(x-cx)+(y-cy)*(y-cy)<=r*r)
            return Math.sqrt(r*r-(x-cx)*(x-cx)-(y-cy)*(y-cy));
        else
            return 0.0;
    }
    if(nfunc2==2){
        if(xmin2<=x && x<=xmin2+1)
            return (Math.sin(Math.PI*(x*2-0.5))+1)/4;
        else
            return 0.0;
    }
    if(nfunc2 == 3){
        if(xmin2<=x && x<=xmin2+1)
            return (Math.sin(Math.PI*(x*2-0.5))+1)/4;
        if(xmax2-1<=x && x<=xmax2)
            return (Math.sin(Math.PI*((x-xmax2+1)*2-0.5))+1)/6;
        else
            return 0.0;
    }
    if(nfunc2 == 4){
        if(x*x+y*y<1)
            return ((Math.cos(Math.PI*x)+Math.cos(Math.PI*y))+1)/2;
        else
            return 0.0;
    }
    if(nfunc2 == 5){

```



```

        if(x*x+y*y<1)
            return ((Math.cos(Math.PI*x)+Math.cos(Math.PI*y))+1)/2;
        else
            return 0.0;
    }
    else
        return 0.0;
}

//初期条件
public double psi(double x,double y){
    double cx = (xmax2+xmin2)/2;
    double cy = (ymax2+ymin2)/2;
    double r = (xmax2-xmin2)/5;

    if(nfunc2==2){
        if(xmin2<=x && x<=xmin2+1)
            return -2*Math.PI*Math.cos(Math.PI*(x*2-0.5))/4;
        else
            return 0.0;
    }
    if(nfunc2 == 3){
        if(xmin2<=x && x<=xmin2+1)
            return -2*Math.PI*Math.cos(Math.PI*(x*2-0.5))/4;
        if(xmax2-1<=x && x<=xmax2)
            return 2*Math.PI*Math.cos(Math.PI*((x-xmax2+1)*2-0.5))/6;
        else
            return 0.0;
    }
    if(nfunc2 == 4){
        if(x*x+y*y<1)
            return (-Math.PI*Math.sin(Math.PI*x)+
                    -Math.PI*Math.sin(Math.PI*y))/2;
        else
            return 0.0;
    }
    if(nfunc2 ==5){
        if(x*x+y*y<1)
            return Math.PI*Math.sin(Math.PI*x)/2;
        else
            return 0.0;
    }
    else
        return 0.0;
}

//初期値
public double phi(double x){
    if(nfunc1 == 0){
        return Math.sin(Math.PI*x);
    }
    if(nfunc1 == 1){
        if(0.375<x && x<=0.5)
            return 8.0*x-3.0;
        if(0.5<x && x<=0.625)
            return -8.0*x+5.0;
        else
            return 0.0;
    }
}

```

```

    }
    if(nfunc1 == 2){
        if(0.0<=x && x<=0.2)
            return (Math.sin(Math.PI*(x*10-0.5))+1)/4;
        else
            return 0.0;
    }
    if(nfunc1 == 3){
        if(0.0<=x && x<=0.2)
            return (Math.sin(Math.PI*(x*10-0.5))+1)/4;
        if(0.9<=x && x<=1.0)
            return (Math.sin(Math.PI*((x-0.9)*20-0.5))+1)/8;
        else
            return 0.0;
    }
    else
        return 0.0;
}

//初期条件
public double psi(double x){
    if(nfunc1 == 0){
        return 0.0;
    }
    if(nfunc1 == 1){
        return 0.0;
    }
    if(nfunc1 == 2){
        if(0.0<=x && x<=0.2)
            return -2.5*Math.PI*Math.cos(Math.PI*(10*x-0.5));
        else
            return 0.0;
    }
    if(nfunc1 == 3){
        if(0.0<=x && x<=0.2)
            return -2.5*Math.PI*Math.cos(Math.PI*(10*x-0.5));
        if(0.9<=x && x<=1.0)
            return 2.5*Math.PI*Math.cos(Math.PI*((x-0.9)*20-0.5));
        else
            return 0.0;
    }
    else
        return 0.0;
}

public void drawAxis(){
    m.move(0.0,0.0); m.draw(1.025,0.0);
    for(double i=0.0;i<=1.0;i+=0.1){
        m.move(i,0.025); m.draw(i,-0.025);
    }
    m.move(0.0,1.05); m.draw(0.0,-1.05);
    for(double i=1.0;i>=-1;i-=0.2){
        m.move(-0.01,i); m.draw(0.01,i);
    }
}
}

//数字しか入らないテキストフィールドの作成
class NumericField extends JTextField{

```

```

public NumericField(String begin){
    super(begin);
    enableEvents(AWTEvent.KEY_EVENT_MASK);
}

public NumericField(String begin,int num){
    super(begin,num);
    enableEvents(AWTEvent.KEY_EVENT_MASK);
}

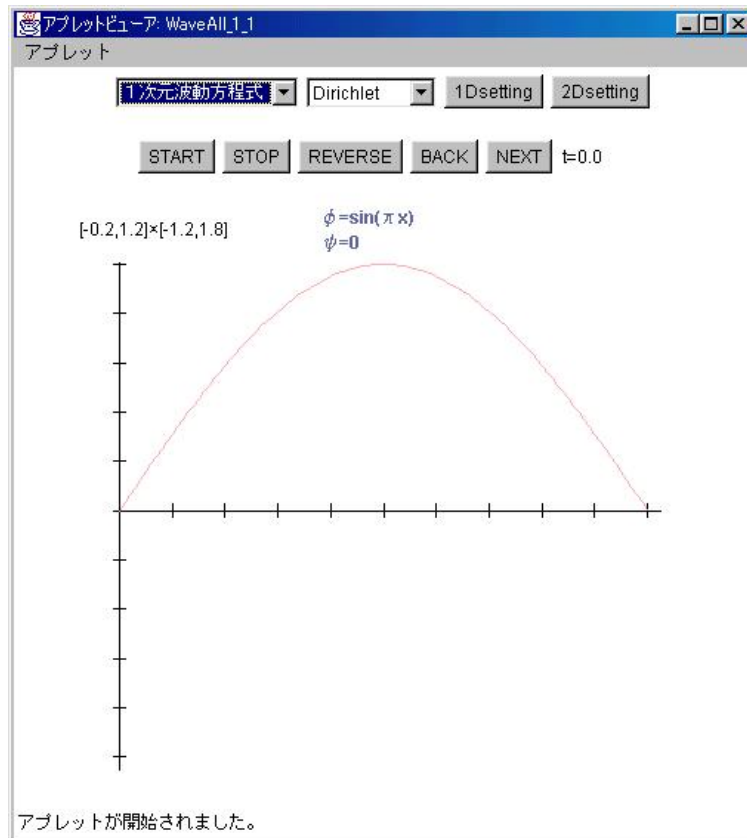
protected void processKeyEvent(KeyEvent e){
    String validValues = "0123456789.- ";
    int code = e.getKeyCode();
    switch(code){
        case 39:break;                //右カーソル
        case 37:break;                //左カーソル
        default:
            char chr = e.getKeyChar();
            if(chr >= ' ' && validValues.indexOf( chr )!=-1){
                return;
            }
            break;
    }
    super.processKeyEvent(e);
}
}

// 2行にまたがるラベル。
class Label2 extends JComponent{
    JLabel row1,row2;

    public Label2(String s1,String s2){
        setLayout(new BoxLayout(this,BoxLayout.Y_AXIS));
        row1 = new JLabel(s1);
        row2 = new JLabel(s2);
        add(row1);
        add(row2);
    }

    public void setText(String s1,String s2){
        row1.setText(s1);
        row2.setText(s2);
    }
}
}

```



1次元波動方程式の設定

? 初期条件を選んでください

任意の振動を繰り返す波

分割数を指定してください(≦100)

100

時間刻みを指定してください

0.01

描写範囲を指定してください 数値

-0.2 1.2 -1.2 1.8

了解 取消し

Java Applet Window

2次元波動方程式の設定

? 初期条件を選んでください

$\phi = \sin(\pi x) + \sin(\pi y), \psi = 0$

分割数  $N_x(= < 50)$   $N_y(= < 50)$   
 40 40

時間刻みを指定してください  
 0.07

描写範囲の指定 数の間には空白を入れて下さい。  
 -2.0 2.5 -2.0 2.5 -1.0 1.0

角度変え (ラジアン) 水平方向 垂直方向  
 0 0

原点移動 (ピクセル) 右向き 上向き  
 0 0

サイズ変え x 軸 y 軸 z 軸  
 250 250 150

了解 取消し

Java Applet Window

## 3.2 プログラムの解説

このプログラムの目的は世界中の人々に波動方程式の数値解析を体感してもらうところにある。そのためにプラットフォームに依存しないという特徴をもつ Java 言語をプログラミング言語として用いた。Java のアプレットという機能を用いれば、アプレットを用いて作ったアプリケーションをそのまま Web 上に公開することが出来るのである。また Web 上での公開をするには値の変更を GUI でしなければならない。その GUI を作るための機能も Java には標準装備されていて楽に GUI を作ることが出来るのである。またその GUI を有効に使うためにマルチスレッドプログラミングを用い、計算の途中でもアニメーションを止めたりすることが出来るようにした。ここに書いてあるプログラムは、それらの機能を駆使したプログラムである。

### 3.2.1 アプレット

Wave1d.java, Wave2d.java でも使ってきたプログラムの形式であるがいったいどんなものなのだろうか？アプレットとは Web ブラウザに表示したり、JDK に付属する appletviewer を使ってウインドウに表示するためのプログラム形式である。アプレット形式で作成したプログラムは、ホームページ内に表示することが出来るのでインターネットを利用して、多くのユーザーがプログラムを実行することが可能になる。

それではアプレットに関するメソッドや注意事項を書こう。

```
import java.applet.Applet;
```

import は、パッケージを使用するための命令である。アプレットを作成する場合には java.applet.Applet を一番最初に指定する。

```
public void init()
```

init メソッドは、アプレットのロード時に最初に実行される。変数名の初期化や GUI のボタン、メニューなどの設定を行うことが多い。

```
public void start()
```

start メソッドは、アプレットの開始時に init メソッドの次に実行する。一度アプレットを表示した後、別のページを参照し、再度戻ってきた時にも実行される。音楽やアニメーションの開始に使うことが多い。

```
public void stop()
```

stop メソッドは、アプレットの停止時に実行される。別のページを参照するとアプレットは一旦停止する。停止する前にこのメソッドが実行される。

```
public void destroy()
```

destroy メソッドは、アプレットのアンロード時に実行される。アンロードはブラウザが終了する時に最後に実行される。

```
public void paint()
```

グラフィックの表示を行うためのメソッドである。アプレットを作成する場合は必ずこのメソッドを定義しなければならない

いろいろなメソッドがあるが単純なアプレットを作成する場合は init メソッドと paint メソッドだけで十分である。

### 3.2.2 マルチスレッドプログラミング

このプログラムの最大の特徴はマルチスレッドプログラミングを行っていることである。スレッドとはコンピューターの処理の流れのことである。少しでもプログラミングをしたことがある人は、コンピューターがプログラムを処理するのに上から順番に処理をしていくことがわかるだろう。これがスレッドである。コンピューターの処理が例外なく一つながりの糸のように繋がっていることから、スレッド（糸）と呼ばれる。マルチスレッドプログラミングとはこのスレッドを2つ以上に増やすことである。要するにいくつもの処理を並行して行えるのである。

例えばGUIなんかはマルチスレッドを使っている。表計算ソフトが複雑な計算を行っている間でも、ユーザーがメニューをクリックするとメニューがさっと表示されますね。これは計算を行っているスレッドとメニューを表示しているスレッドが独立に働いているのからなのだ。これがもし一つのスレッドしかなかったら計算が終わるまでメニューを開くことは出来ない。このようにマルチスレッドプログラミングは私たちが普段あたりまえと思っているようなことに使われているのだ。

僕の今回のプログラミングではアニメーションを描き続けるスレッドとボタン操作のスレッドは別になっている。そうすることによって、アニメーションを描きつつづけている最中でも一時停止をさせたり、初期値の関数を変えたりする。もしスレッドが一つしかなかったら、描き終わるまで何も操作が出来ない。

それではどうやってこのマルチスレッドプログラミングを行うのでしょうか？C言語とは違いJavaには最初からマルチスレッドプログラムが前提となっているため特別なライブラリーを組み込む必要は無い。それどころかプログラミング自体も普通のプログラムを書くのとあまり変わらないように記述できるのである。

Thread(スレッド)機能を利用するにはThreadクラスを継承する方法とRunnableクラスをインターフェースする方法とがある。このプログラミングでは後者のRunnableクラスをインターフェースしている。理由はJavaの継承の機能の制限からきている。Javaでは一つのクラスしか継承できないという制限がある。このプログラムは見てもらえばわかると思うがAppletを継承している。それ故に必然的に後者の方法になる。使い方は両者ともそんなに違いは無い。今回は後者の方法について説明させていただく。

- スレッドを行いたいクラスに `Runnable` を implements ( インターフェースの方法 ) する。
- `Runnable` クラスをインターフェースしたクラスに `run()` メソッドを書き、それを実装する。( スレッドは `run()` でくくられた部分しか動かないから、並行して行いたい処理をこの部分に書く。
- `Thread` クラスのインスタンスを生成する。( 例 `Thread th = new Thread();` )
- `Thread` のインスタンスメソッドである `start()` をスレッドを行ったクラスのインスタンスを引数として実行する。( クラス `Ex` 内に `run()` を実装している時の例 `th.start(new Ex());` )

### 3.2.3 ダブルバッファリング

複雑なイメージの場合や作図するのに時間がかかる場合は、「イメージの作図」に時間がかかる。このような処理ではアニメーションを行うと作図の過程が見えてしまう。

ダブルバッファリングは作図用のバッファに作図し、作図が終了した段階で一度にバッファを表示する手法である。

さらに今回のプログラムではもう一つの利点がある。ディスプレイに表示される画面に何かしら書くときは必ず `paint(Graphics g)` のなかでしなければならない。しかし作図用のバッファなら `paint()` 以外の部分でも描くことが出来る。すなわち `run()` のなかで図を描くことができ、そして `paint()` 内でそのバッファをくっつけるだけでいいのだ。

初めの `//もう一つの画面の定義` とはその作図用のバッファの定義のことである。

### 3.2.4 各メソッド

注意の要る変数

プログラムを読んでいると `re_t` と `im_t` の二つがあることに気づくだろう。これの読み方はリアルティーとイメージティー。その名の通り本当の `t` と嘘の `t` だ。本当の `t` とは波が動き出してから経過した時間を求めるための係数。すなわち 1 次元で例えると  $t = j\tau$  の  $j$  にあたるのが `re_t` である。これは `reverse` ボタンを押されれば値は逆流するし、`back` ボタンや `next` ボタンを押しても変化するこの  $j$  にあたる物の名前が欲しいのでここでは時間数と呼ぶことにする。一方 `im_t` は最初に 0、次に 1、次 2 ... と何を押そうがひたすら増えつづけていく。

これには深い理由がある。それは `reverse` ボタンと差分を求める `f1()`, `f2()` の二つの存在にある。このメソッドを見てもらうとわかると思うのだがこの二つのメソッドは時間数を引数にして動いている。そしてこの時間数が 0 の時、1 の時、それ以外の正数の時の 3 つに分けられている。そしてこの場合わけが曲者なのである。つまり一番最初に時間数 0 の時と 1 の時を呼び出したら、その後永遠にこの二つの所 ( 時間数 0 と 1 ) には行きたくないのである。

例えば波を動かし始めると、時間数がどんどん増えていく。そこで `reverse` ボタンを押し、時間を逆流させるとこの時間数も減っていくわけである。そして困ったことにこの時間数がまた 1 になった時にメソッド `f1()`, `f2()` において時間数が 1 の所に行かれると値が変わってくる



のである。ここで時間数が実際に 1 でも  $f1()$ ,  $f2()$  の場合わけで 2 以上のところにいってもらえればうまくいけるのである。

いろいろ考えた挙句に思いついたのがこの方法である。要は実際の時間数を引数に渡すもんだから値が変わってくる。だったら違う物を引数として渡してしまえということで考えたのがこの  $re\_t, im\_t$  である。この実際の時間数を  $re\_t$  で表して実際の時間数とは別な物を  $im\_t$  で表しているのである。だからメソッド  $f1()$ ,  $f2()$  を呼ぶ際にはこの  $im\_t$  を引数として呼んでいる。

#### `public void init()`

アプリットに関するメソッドの一つである。ここではさまざまな初期設定を行っている。さまざまなボタンの設定やチョイスの設定から作図用画面の作成や MitsuiWorld の設定まで多種多様の初期設定をここで行っている。しかし気おつけなければならないことが 1 つある。それはファイルを呼び出された時に、一番最初に一回だけ呼び出されるということだ。すなわち GUI で値を変えたい物をここでしか設定していないと、いくら値を変えても変わらないのである。GUI によって値を変えたい物はほかでも定義した方がよい。ここで設定するのは初期値だけにとどめておいた方がよい。そのため値を変更する場所を `paint` 内でしている。

#### `public void start()`

アプリットに関するメソッドの一つである。ここでの目的はスレッドを走らせる事だけである。いつ呼び出されるかなどは上を参照して欲しい。そうすればなぜここでスレッドをスタートさせるのかがわかると思う。

#### `public void run()`

マルチスレッドを行うにあたって実装しなければならないメソッド。この `run()` のなかでひたすらアニメーションを描きつづけている。

`while(true)` で `run` 全体をかこっているのでアニメーションを描くスレッドはひたすら走り続ける。

動作を表す `runmove` が `true` の時は画面に描きます。`false` の時はスレッドがとまっている時である。`start` ボタンを押すことによって状態を `true` にしスレッドを動き出させ、`stop` ボタンを押すことによって状態を `false` にしスレッドを一時停止させる。

その他のところでは 1 次元の時と 2 次元の時とでわかれてはいるものの、要は作図用画面に作図しているのである。この部分が作図している中心部である。。

#### `public void stop`

スレッドの存在を消している。ここでスレッドを消すことによって余計な計算をさせないで済む。

public void actionPerformed(ActionEvent e)

この部分で全てのボタンを押された時のアクションを記述している。  
e.getSource()==\*\*\*\*で何を押したか判断する。

start ボタンを押した時

スレッドを動かすためにこのメソッドを作った。runmove=true にし、while(!runmove) に閉じ込められていればそこから脱出し、スレッドを活動させ、アニメーションを描きはじめる。while(!runmove) に閉じ込められていなければ何もおこらない。

stop ボタンを押した時

スレッドを一時停止させる目的で作った。runmove=false にし、while(!runmove) に閉じ込める。一時停止のとき、すなわち runmove=false のときは、初期状態にする。

back ボタンを押した時

一時停止のとき、すなわち runmove=false のときにのみ動作する。1 コマ戻す動作をする。波動方程式は熱方程式とは違い逆流できる。 $j \geq 1$  のときの 1 次元波動方程式の差分解を見ると

$$U_i^{j+1} = 2(1 - \lambda^2)U_i^j + \lambda^2(U_{i+1}^j + U_{i-1}^j) - U_i^{j-1}$$

である。これを少し変形させると

$$U_i^{j-1} = 2(1 - \lambda^2)U_i^j + \lambda^2(U_{i+1}^j + U_{i-1}^j) - U_i^{j+1}$$

となる。ここでプログラムに戻って考えてみよう。1 次元波動方程式で時刻  $j$  で stop ボタンを押したそきの  $u1, u2, u3$  を考えると

$$\begin{pmatrix} u1 \\ u2 \\ u3 \end{pmatrix} = \begin{pmatrix} U_i^{j-1} \\ U_i^j \\ U_i^j \end{pmatrix}$$

これを無限回コマを巻き戻しが出来て、なおかつ start ボタンをおしたらそこからスタートするようにするには、1 コマ戻した時に  $u1, u2, u3$  は

$$\begin{pmatrix} u1 \\ u2 \\ u3 \end{pmatrix} = \begin{pmatrix} U_i^{j-2} \\ U_i^{j-1} \\ U_i^{j-1} \end{pmatrix}$$

となる必要がある。それを考えながら back ボタンが押された時の動作を見てみよう。1 次元波動方程式で時刻  $j$  で stop ボタンを押したそきの  $u1, u2, u3$  を考えると先ほどの状態だったが、最初の for 文で  $u1, u2, u3$  は

$$\begin{pmatrix} u1 \\ u2 \\ u3 \end{pmatrix} = \begin{pmatrix} U_i^j \\ U_i^{j-1} \\ U_i^j \end{pmatrix}$$

となり `double[] u = f1(t);` とすることにより  $u_1, u_2, u_3$  は

$$\begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} U_i^{j-1} \\ U_i^{j-2} \\ U_i^{j-2} \end{pmatrix}$$

となり、次の for ぶんで

$$\begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} U_i^{j-2} \\ U_i^{j-1} \\ U_i^{j-1} \end{pmatrix}$$

としている。これで上の条件を満たす  $u_1, u_2, u_3$  が出来たことになる。そしてここきて  $u_3$  すなわち  $U_i^{j-1}$  を描いている。2次元もアルゴリズムは全く同じである。

next ボタンを押した時

一時停止のとき、すなわち `runmove=false` のときにのみ動作しする。1コマ進める動作をする。 $t$  の値を1つ増やし `double[] u = f1(t)` または `double[][] u = f2(t)` をすることによって一足お先に次の値をもらってそれを描いているだけのかんたんなアルゴリズムである。

reverse ボタンを押した時

先に書いた back と同じように波動方程式の逆流の特性をつかう。この部分では配列を変えるだけである。描く部分は `run()` に託している。つまりこの部分では  $u_1, u_2, u_3$  の値の持ち方を `run()` にあわせる必要があるのである。1次元波動方程式で時刻  $j$  時の  $u_1, u_2, u_3$  を考えると

$$\begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} U_i^{j-1} \\ U_i^j \\ U_i^j \end{pmatrix}$$

である。この状態で  $U_i^j$  を描いているのである。このボタンを押してから `run()` が  $U_i^{j-1}$  描いて逆流するためには

$$\begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} U_i^{j+1} \\ U_i^j \\ * \end{pmatrix}$$

となる必要がある。ただし\*は何でも良い。それをふまえてプログラムを見てみよう。最初に

`double[] u = f1(t)` とすることによって時刻  $j$  の時の  $u_1, u_2, u_3$  を時刻  $j+1$  の時の  $u_1, u_2, u_3$  にしていい。すなわち今  $u_1, u_2, u_3$  は

$$\begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} U_i^j \\ U_i^{j+1} \\ U_i^{j+1} \end{pmatrix}$$

となっている。そこで次の for 文によって  $u_1, u_2, u_3$  は

$$\begin{pmatrix} u1 \\ u2 \\ u3 \end{pmatrix} = \begin{pmatrix} U_i^{j+1} \\ U_i^j \\ U_i^j \end{pmatrix}$$

となっている。これは上の条件を満たす。ここで上では\*だった部分をわざわざ  $U_i^j$  にしたのはデバッグのためである。reverse だけだったらいいのだが他のボタンのことも考えてこうした。後は run() がやってくれる。

### 1Dsetting,2Dsetting ボタンを押した時

これを押すことによってダイアログを呼び出している。そしてダイアログ上で詳細の設定ができるようにした。ここの部分はごちゃごちゃ長くて難しそうだがやってることはものすごい単純である。始めの部分でラベルならラベル、テキストならテキストのオブジェクトをひたすら作っているのである。それを

```
Object[] obj = {s,c_func2,pdivide,inputTau,inputArea,pangle,porigin,plength};
```

として一つのオブジェクトにまとめダイアログにぶち込むだけである。

そして何かを選んだら初期状態にするようにした。

```
public void itemStateChanged(ItemEvent e)
```

ここで選択式だった GUI の動作を決定している。\*.getSelectedIndex() で何を選んだか番号で表している。表し方は上から順に 0, 1, ... となる。だから次元を表すときに 1 次元波動方程式を dimension=0、2 次元波動方程式を dimension=1 としたのである。

そして何かを選んだら初期状態にするようにした。

```
public void paint(Graphics g)
```

アプレットに関するメソッドである。ここのメインの動作はただ一つ。今まで作図用の画面に描いてきたものをここで画面に貼り付けるだけ。

ただごちゃごちゃして見えるのは GUI で設定を変えた物をここで再定義したりしているのだ

ただ、 $ret = 0$  のときの描写をここでしている。そうすることによって初期状態に戻した時に  $ret = 0$  の画面が描かれからである。

```
public double[][] f2(int t)
```

ここでは Wave2d.java でいうと paint() 内でしていた計算を関数として扱っただけである。要はこの部分で 2 次元波動方程式の差分解を計算しているのである。ここは今までと同じなので特筆はしない。

```
public double[] f1(int t)
```

ここでは Wave1d.java でいうと paint() 内でしていた計算を関数として扱っただけである。要はこの部分で 1 次元波動方程式の差分解を計算しているのである。ここは今までと同じなので特筆はしない。

```
public double phi()
```

この部分で 1 次元と 2 次元の初期値をあつかっている。同じ関数名で 1 次元と 2 次元を両方扱えるのはオーバーロードという Java の特徴の一つでもある。式を変えることによってさまざまな波を描く事が出来る。

```
public double psi()
```

この部分で 1 次元と 2 次元の初期条件を扱っている。ここもオーバーロードを使っている。この式を変えることによって phi() で描いた波の動きや状態を変えることが出来る。

```
public void drawAxis()
```

ここで 1 次元波動方程式の軸をえがいている。

```
class NumericField extends JTextField
```

```
enableEvents(AWTEvent.KEY_EVENT_MASK);
```

コンストラクタないでこれを書く。そうするとこのオブジェクトの中でキーイベントを処理するという宣言になる。

```
protected void processKeyEvent(KeyEvent e)
```

キーイベントが発生するとこのメソッドが呼ばれる。

```
String validValues = "0123456789.-";
```

有効文字を入れる

```
int code = e.getKeyCode();char chr = e.getKeyChar();
```

文字コードを取り出す。

```
if(chr >= ' ' && validValues.indexOf( chr )== -1)return;
```

有効数字でないイベント処理はここで中断する。

```
super.processKeyEvent(e);
```

有効数字はここでスーパークラスである JTextField に文字を渡され、通常に処理される。

```
class Label2 extends JComponent
```

これはそんなに難しくは無いと思う。ただ2つのラベルを縦に並べそれを1つのコンポーネントとして扱っているだけだからだ。ほんとにそれだけ。

## 第4章 MitsuiWorldのドキュメンテーション

この卒論を読んだ人は MitsuiWorld とはいったいなんだ？と疑問に思ったことだろう。この章ではその MitsuiWorld について少し説明しよう。MitsuiWorld とはその名の通りこの卒論を書いている僕自身が作り上げたライブラリーである。何に使うのかというと、みなさんもお気づきのように Java において数学の関数を描くのには結構役に立つのではないかなーと思っている。しかし出来の方に満足はしていない。もう少し時間があればもっと勉強してもっと良いものが作れたと思う。何に一番悔やんでいるのかというと、3Dを描くのに Java3D のパッケージを使わなかったことだ。Java 自身にそのような機能がついているにもかかわらず、それを使わなかった。いや、正確に言うと使えなかった。使えなかった最大の理由。それは Java3D のドキュメントが英語で書かれているのしか見つけれなかったからである。英語が得意ではない私としてはさすがにこれには手がつけられなかった。そのような理由で満足のいかない作品を提出することになったことをここで深くおわびします。しかしその割にはたいしたものが出来たなと思っている一面もある。だからこれから将来、桂田研の生徒に使われることがあればうれしく思う。

## 4.1 MitsuiWorld の一覧表

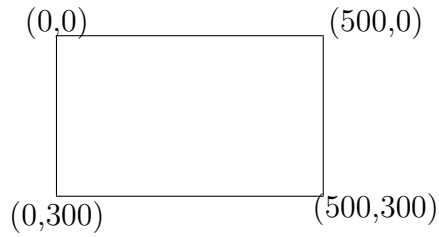
共通のメソッド	1.setGraphics	今から描く Graphics をセットするメソッド
	2.setScreenSize	描写画面のサイズをセットするメソッド。
	3.setColor	線の色を決めるメソッド
2次元メソッド	4.setArea	描写範囲を決めるメソッド
	5.move	カーソルの位置を動かすメソッド
	6.draw	線をひくメソッド
3次元の一般のメソッド	7.setArea2	描写範囲を決めるメソッド
	8.move2	カーソルの位置を動かすメソッド
	9.draw2	線をひくメソッド
3次元の鳥瞰図メソッド	10.drawBirdView	鳥瞰図を描く（線のみ）
	11.fillBirdView	鳥瞰図を描く（色つきだが奥が透けて見える）
	12.hideBirdView	鳥瞰図を描く（色つきで陰面消去してある）
3次元の視覚変化メソッド	13.changeAngle	視覚の角度を変えるメソッド
	14.changePosition	原点の位置を変えるメソッド
	15.changeSize	枠の長さを変えるメソッド
	16.changeReflect	透視投影の比率を変えるメソッド
	17.drawAxis2	枠を描くメソッド
番外メソッド	18.drawHideView	鳥瞰図を描く（色つきで陰面消去してある）

## 4.2 座標系のお話

ここで少し座標系についてのお話をしよう。プログラムを書いたことが事がある人ならばすでに気づいていると思うが、プログラムで数学のグラフを描こうとすると結構やっかいなのである。なぜならばプログラムが持っている座標と数学の座標は全く違うのである。プログラムが持っている座標とは左上角が(0, 0)であって、右下に行くにしたがって値が大きくなるのである。正確に言うと右に行けば行くほどxの座標があがり、下に行けば行くほどyの座標が上がるのである。それに比べて数学の座標とは絶対的な基準となる点が無いのである。原点という物は存在するがそれをグラフ内に描かなければならないということも無いし、原点を何処かに固定しなければならないということも無いし、1マスの長さを何ミりにしなければならないという決まりも無い。

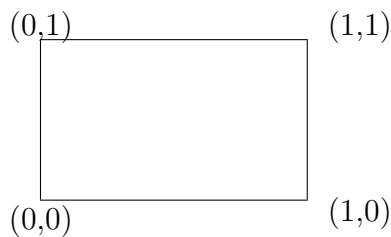
プログラムではピクセルという一つのきまったマスが用意されていて、それで指定するのである。例えばある画面を横に500ピクセル縦に300ピクセルと指定すると、横に長さ500ピクセル、縦に長さ300ピクセルの画面ができ、また下の図のようにそのまま座標になるのである。



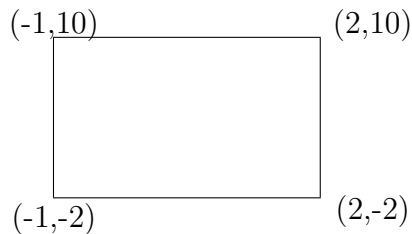


このようにピクセルで指定する方法をこれからピクセル指定と呼ぶ。

さて、それではこの画面に数学のグラフを描いてみよう。原点の位置をどこにしようかな？ 左上角？ 左下角？ 右上角？ 右下角？ 中央？ といくらでも考えられる。正解はどこでもいいのである。例えば  $[0,1] \times [0,1]$  のグラフが見たいと指定すると上のグラフは



という風に変換することも可能である。また  $[-1,2] \times [-2,10]$  のグラフが見たいと指定するとグラフは



という風に座標を変換することもできる。このように座標を変換してしまうと数学のグラフを描くのに大変楽である。このように変換した座標でいろいろ指定することをこれから数学座標指定と呼ぶ。

### 4.3 MitsuiWorld の使い方

まず以下のようにして MitsuiWorld のインスタンスを生成する。

```
MitsuiWorld MitsuiWorld_obj = new MitsuiWorld();
```

これは MitsuiWorld に関わらず全てのクラスのインスタンスを生成するときと同じように生成する。

### 4.3.1 2次元3次元共通のメソッド

#### メソッドその1 setGraphics

このメソッドは今から描く Graphics をセットする関数である。すなわち draw や draw2 などのメソッドで描きたい画面の Graphics をセットするのである。

使い方

```
MitsuiWorld_obj.setGraphics(Graphics_obj);
```

#### メソッドその2 setScreenSize

これは setGraphics でセットした Graphics によって描かれる画面のサイズをセットするメソッドである。一つ気をつけて欲しいのは画面のサイズをこうするぞと決めるのではなくて、Graphics によって描かれる画面のサイズが既に決まっていると思うから、その値をここに入れるのである。すなわち、このメソッドは setGraphics を呼び出した時点で何を入れなければいけないか決まるのである。

別にそうしないとバグが起こるわけではない。ただ Graphics によって描かれる画面のサイズを入れることによって、画面にあった（大きさがちょうどよい）図を描くことができるのである。だからもし、Graphics によって描かれる画面のサイズより小さい値を入れれば描いた図の周りに空白の画面ができるし、そのサイズより大きければ画面から図がはみ出してしまうだけである。

ピクセル指定

使い方

```
MitsuiWorld_obj.setScreenSize(intwidth, intheight);
```

#### メソッドその3 setColor

このメソッドは描く線の色を決めるメソッドである。番号で指定と直接の指定（カラーオブジェクトを入れる）の2通りの指定の仕方がある。以下にカラー番号を記述する。カラー番号12を越えると黒いろになる。

Color クラスの変数	色	色番号
Color.black	黒	0
Color.white	白	1
Color.blue	青	2
Color.red	赤	3
Color.yellow	黄	4
Color.green	緑	5
Color.darkGray	暗い灰色	6
Color.gray	灰色	7
Color.lightGray	明るい灰色	8
Color.pink	ピンク	9
Color.orange	オレンジ	10
Color.magenta	マゼンタ（赤紫）	11
Color.cyan	シアン（青紫）	12

使い方

```
MitsuiWorld_obj.setColor(Color_obj);
```

または

```
MitsuiWorld_obj.setColor(intcolorNumber);
```

プログラムで setColor を番号で色を指定しているものをほとんど見ないと思った人もいるかもしれない。ただこの番号指定のメソッドを作ったのがレポート提出間際だったのでこのレポートの中のを換えるまでは時間がなかったので、昔ながらの記述のまま提出している。だから安心して色を番号指定してもらって良い。

### 4.3.2 2次元に関するメソッド

#### メソッドその4 setArea

このメソッドは二次元グラフを描くときの画面に描写する範囲を決める。 $[a, b] \times [c, d]$  となるように書き込む。

数学座標での指定。

使い方

```
MitsuiWorld_obj.setArea(double a, double b, double c, double d);
```

#### メソッドその5 move

このメソッドはカーソルの位置を指定した場所に移動させる。カーソルとはグラフ上の現在地を示しているポイントである。線を描く時などはこのポイントから描き始める。

数学座標での指定。

使い方

```
MitsuiWorld_obj.move(double x, double y);
```

## メソッドその6 draw

このメソッドは現在あるカーソルの位置から指定した場所まで線をひく。そしてカーソルの位置を線をひき終わった場所まで持っていく。

数学座標での指定。

使い方

*MitsuiWorld obj.draw(double x, double y);*

今までの6つのメソッドを使うことによって二次元グラフを描けるようになった。そこで次は今までの6つのメソッドを用いた具体的な例を試みる。

例1 applet でx軸y軸を描きます。そして赤色で  $\sin(\pi x)$  のグラフを描きます。

```
import java.applet.Applet;
import java.awt.*;
import Mitsui.*;

//パッケージを呼び込む

public class Ex1 extends Applet{
    public void paint(Graphics g){ //applet の paint 部分
        MitsuiWorld_2 m = new MitsuiWorld_2(); //インスタンス生成
        m.setGraphics(g);
        m.setScreenSize(getSize().width,getSize().height); //描く画面をセット
        // [ a , b ] × [ c , d ]
        m.setArea(-1.0,1.0,-1.0,1.0); //描写範囲を決める

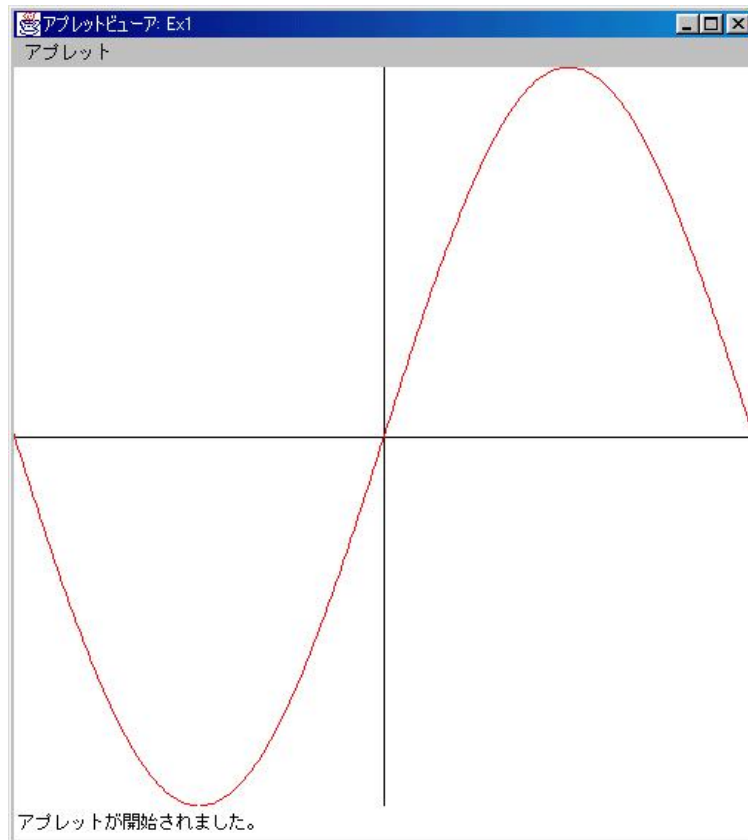
        //座標軸を設定
        m.move(-1.0,0.0); m.draw(1.0,0.0);
        m.move(0.0,1.0); m.draw(0.0,-1.0);

        m.setColor(Color.red); //色を赤にする

        double h=0.01; //sin( x) を 0.01 刻みで描く
        double x=-1.0;
        m.move(x,f(x));
        for( x=-1.01;x<=1.0;x+=h){
            m.draw(x,f(x));
        }
    }

    public double f(double x){
        return Math.sin(Math.PI*x);
    }
}
```

- 注1 paint の引数を abc としたら setGraphics(abc) とする。  
すなわち public void paint(abc) とすると setGraphics(abc); とする。
- 注2 setArea メソッドは setScreenSize を定義してから定義すること。
- 注3 import Mitsui.\*; の部分は MitsuiWorld をパッケージとして使わなければ別に書かなくても良い。



### 4.3.3 3次元に関するメソッド

それでは3次元グラフを描くためのメソッドの使い方を説明しよう。

## 一般のメソッド

### メソッドその7 setSArea2

これは2次元グラフでいう setArea と同じ役割で描写範囲を決める。2次元と違う点はただ一つ、z座標も指定すること。 $[a, b] \times [c, d] \times [e, f]$ となるように書き込むこと。このメソッドを呼び出すことによって自動的に周りの枠が描かれる。

数学座標での指定。

使い方

```
MitsuiWorld_obj.set2(double a, double b,
                      double c, double d,
                      double e, double f);
```

## その8 その9 move2,draw2

これは2次元グラフで言う move や draw と全く同じである。上同様違うのは z 座標まで指定できることである。

数学座標での指定。

使い方

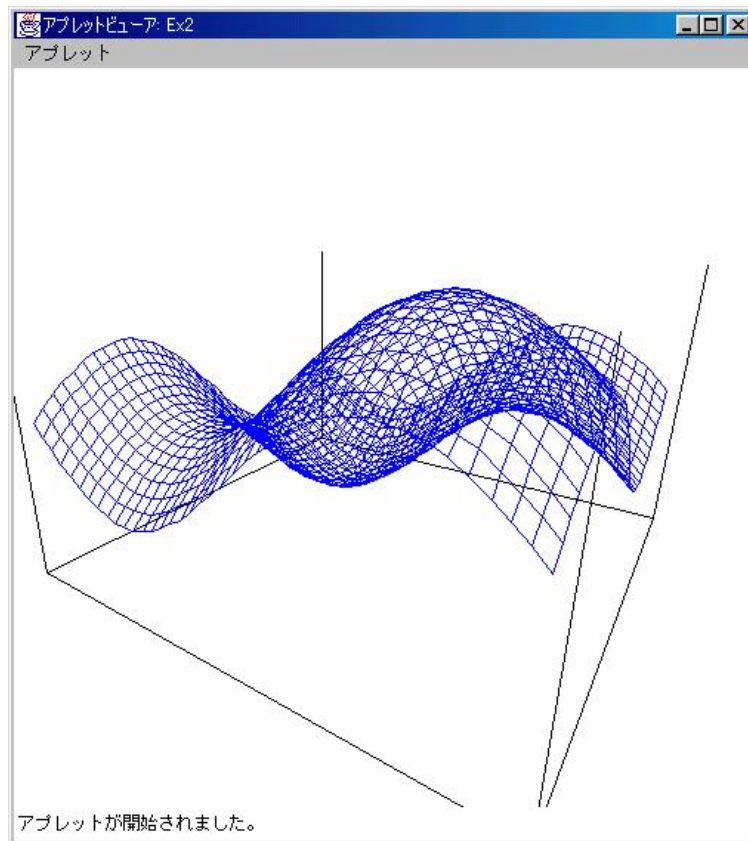
```
MitsuiWorld_obj.move2(double x, double y, double z);  
MitsuiWorld_obj.draw2(double x, double y, double z);
```

ここまでで基本的な描き方は出来るようになる。そこで今までのメソッド(7、8、9)で具体的な例をやってみる。

例2 applet で  $\sin(\pi x) + \sin(\pi y)$  のグラフを描きます。

```
import java.applet.Applet;  
import java.awt.*;  
import Mitsui.*;  
  
public class Ex2 extends Applet{  
    public void paint(Graphics g){  
        MitsuiWorld_2 m = new MitsuiWorld_2();           //インスタンス生成  
        m.setGraphics(g);  
        m.setScreenSize(getSize().width,getSize().height); //描く画面をセット  
        // [ a , b ] × [ c , d ] × [ e , f ]  
        double xmin=-1.0;  
        double xmax=1.0;  
        double ymin=-1.0;  
        double ymax=1.0;  
        double zmin=-2.0;  
        double zmax=2.0;  
        m.setArea2(xmin,xmax,ymin,ymax,zmin,zmax);        //描写範囲を決める  
  
        //この時点で枠が描かれているから2次元の時みたいに軸を描く必要はない。  
  
        m.setColor(Color.blue);                            //色を青にする  
  
        for(double x=xmin;x<=xmax;x+=0.05){                //y 軸に平行な線を描く  
            m.move2(x,ymin,f(x,ymin));  
            for(double y=ymin;y<=ymax;y+=0.05){  
                m.draw2(x,y,f(x,y));  
            }  
        }  
        for(double y=ymin;y<=ymax;y+=0.05){                //x 軸に平行な線を描く  
            m.move2(xmin,y,f(xmin,y));  
            for(double x=xmin;x<=xmax;x+=0.05){  
                m.draw2(x,y,f(x,y));  
            }  
        }  
    }  
  
    public double f(double x,double y){  
        return Math.sin(Math.PI*x)+Math.sin(Math.PI*y);  
    }  
}
```

注 例 1 と同じことがいえます。



## BirdView メソッド

### メソッドその 1 0 drawBirdView

これは例 2 でしてくれたことを自動的にしてくれるメソッドだ。drawBirdView( 2 次元配列 ,  $x$  の分割数 ,  $y$  の分割数 , 線をひく間隔) という風に入力する線をひく間隔というのは、例えば分割数 1 0 0 で間隔数 1 だと 1 0 0 本線をひく。これを間隔数 2 にすると線の本数は 5 0 本になるのだ。すなわち  $N_x, N_y$  を二つとも分割数 5 0 にして drawBirdView( $u, N_x, N_y, 1$ ) とするのと、 $N_x, N_y$  を二つとも分割数 1 0 0 にして drawBirdView( $u, N_x, N_y, 2$ ) とするのでは線の本数は同じになるのだ。しかし分割数 1 0 0 の方がきれいに書ける。

使い方

```
MitsuiWorld_obj.drawBirdView(double[][] u, int nx, int ny, int h);
```

### メソッドその 1 1 fillBirdView

これは drawBirdView の色つきバージョンだ。しかし奥が透けて見える。使い方は drawBirdView と全く同じ。分割数を増やせば増やすほどきれいに描ける。

## メソッドその 1 2 hideBirdView

これは fillBirdView の奥が透けて見えないバージョン。使い方は全く同じ。これに限って分割数 1 0 0 で間隔数 2 と分割数 5 0 で間隔数 1 なのと全く同じである。ただ使い方をそろえただけなのだ。だから分割数をそこそこに。

例 3 それではここで例 2 と全く同じものを drawBirdView で描いてみよう。

```
import java.applet.Applet;
import java.awt.*;
import Mitsui.*;

public class Ex4 extends Applet{
    public void paint(Graphics g){
        MitsuiWorld_2 m = new MitsuiWorld_2();           //インスタンス生成
        m.setGraphics(g);
        m.setScreenSize(getSize().width,getSize().height); //描く画面をセット
        // [ a , b ] × [ c , d ] × [ e , f ]
        double xmin=-1.0;
        double xmax=1.0;
        double ymin=-1.0;
        double ymax=1.0;
        double zmin=-2.0;
        double zmax=2.0;
        m.setArea2(xmin,xmax,ymin,ymax,zmin,zmax);       //描写範囲を決める

        //この時点で枠が描かれているから 2 次元の時みたいに軸を描く必要はない。

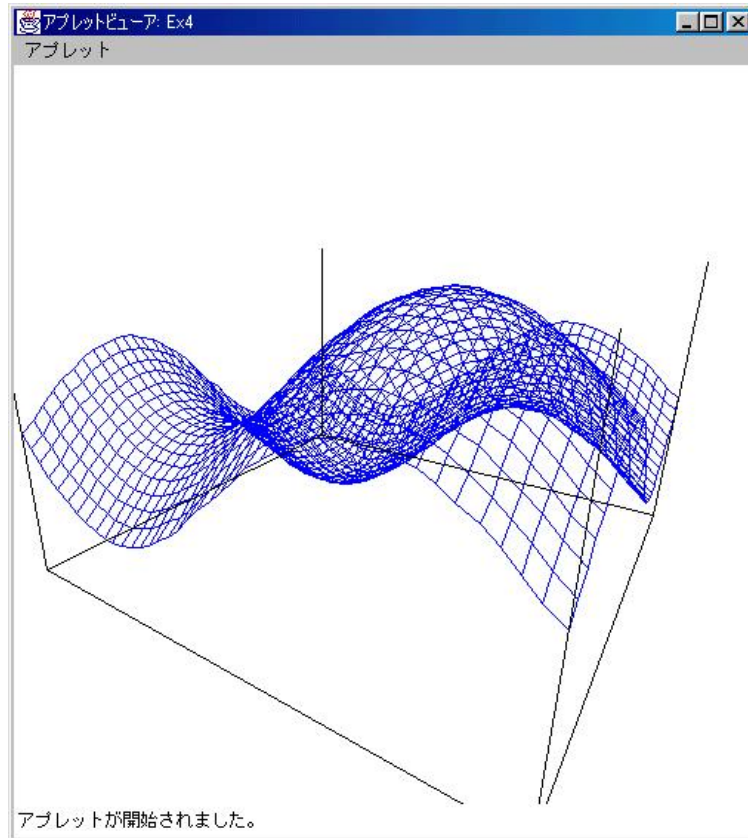
        m.setColor(Color.blue);                          //色を青にする

        //次からが Ex2 と違うところ。
        int nx = 40;
        int ny = 40;
        double[][] u = new double[nx+1][ny+1];
        double dx = 0.05;
        double dy = 0.05;
        for(int i=0;i<=nx;i++){                           //配列に f(x,y) の値をいれる。
            double x=xmin+i*dx;
            for(int j=0;j<=ny;j++){
                double y=ymin+j*dy;
                u[i][j] = f(x,y);
            }
        }

        m.drawBirdView(u,nx,ny,1);
    }

    public double f(double x,double y){
        return Math.sin(Math.PI*x)+Math.sin(Math.PI*y);
    }
}
```





## 視覚変化のメソッド

今までの MitsuiWorld の基本的な使い方は全てである。次からは視覚の角度を変えたりといった知らなくても大丈夫なものだ。しかし知っていればかなり有効なメソッドである。

### メソッドその 1 3 changeAngle

これは 3 次元グラフの視覚の角度を変えるものである。changeAngle(水平方向, 垂直方向) といった感じだ。水平方向は記入した角度分右の方向から、垂直方向は記入した角度分上の方からグラフを見ている感じになる。

ラジアン指定

使い方

```
MitsuiWorld_obj.chngeAng(int x, int y);
```

### メソッドその 1 4 changePosi

これは 3 次元グラフを描く場所を変える。changePosition(横方向, 縦方向) でしている。横方向に指定した分だけグラフ全体が右に移動し、縦方向に指定した分だけグラフ全体が上に移動する。左もしくは下に移動させたかったらマイナスをつけるように。

ピクセル指定

使い方

*MitsuiWorld\_obj.chngePosi(int x,int y);*

## メソッドその 15 changeSize

これは 3 次元グラフの軸の長さを決める。changeSize(x 軸の長さ, y 軸の長さ, z 軸の長さ) という感じである。軸の長さとは最小値から最大値までのピクセルの長さである。例えば長さが 250 ピクセルの時、x の範囲が [0,1] だとすると 0 から 1 までの長さは 250 ピクセルだし、範囲が [0,5] だとすると 0 から 1 までの長さは 50 ピクセルである。軸の長さを長くしたかったら+\*\*とし、短くしたかったら-\*\*とする。

ピクセル指定。

使い方

*MitsuiWorld\_obj.changeSize(int x,int y,int z);*

## メソッドその 16 changeReflect

これは 3 次元のグラフを作成するときに透視投影という方法で作成しているのだが、その比率を変えるのに使う。これは直接値を打ち込むこと。初期値は  $S = 400, d = 500$  である。よっぽどのことがない限り使わない関数だ。(自分でもない) むしろ無くてもいい。

使い方

*MitsuiWorld\_name.changeReflect(int s,int d);*

例 4 それではここで例 2 をメソッド ( 1 3 , 1 4 , 1 5 ) をつかって視覚を変えてみよう。

```
import java.applet.Applet;
import java.awt.*;
import Mitsui.*;

public class Ex3 extends Applet{
    public void paint(Graphics g){
        MitsuiWorld_2 m = new MitsuiWorld_2();           //インスタンス生成
        m.setGraphics(g);
        m.setScreenSize(getSize().width,getSize().height); //描く画面をセット
        //以下 3 行が例 2 と違うところ

        //今より右回りに 0 度、上方向に 3 0 度上げる
        m.changeAngle(0,30);
        //今より x 軸方向に 0 ピクセル y 軸方向に 1 5 0 ピクセル移動させる。
        m.changePosition(0,150);
        //x 軸を 2 0 0、y 軸を 2 0 0、z 軸を 1 0 0 にする。
        m.changeSize(-50,-50,-50);

        // [ a , b ] × [ c , d ] × [ e , f ]
        double xmin=-1.0;
        double xmax=1.0;
        double ymin=-1.0;
        double ymax=1.0;
        double zmin=-2.0;
        double zmax=2.0;
        m.setArea2(xmin,xmax,ymin,ymax,zmin,zmax);        //描写範囲を決める
    }
}
```

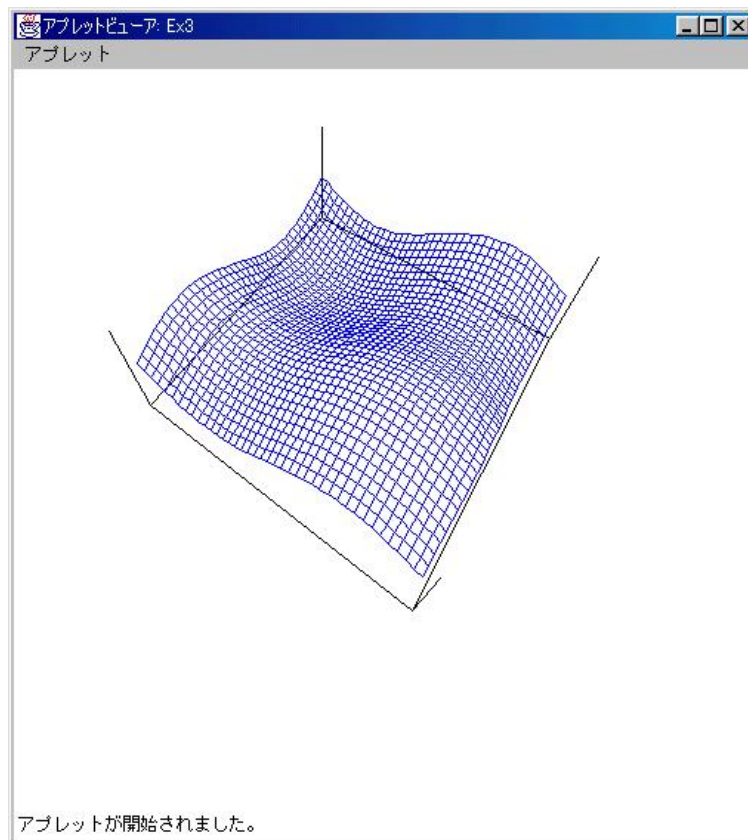
//この時点で枠が描かれているから 2 次元の時みたいに軸を描く必要はない。

```
m.setColor(Color.blue); //色を青にする

for(double x=xmin;x<=xmax;x+=0.05){ // y 軸に平行な線を描く
    m.move2(x,ymin,f(x,ymin));
    for(double y=ymin;y<=ymax;y+=0.05){
        m.draw2(x,y,f(x,y));
    }
}
for(double y=ymin;y<=ymax;y+=0.05){ // x 軸に平行な線を描く
    m.move2(xmin,y,f(xmin,y));
    for(double x=xmin;x<=xmax;x+=0.05){
        m.draw2(x,y,f(x,y));
    }
}
}

public double f(double x,double y){
    return Math.sin(Math.PI*x)+Math.sin(Math.PI*y);
}
}
```

注 メソッド 13 ~ 15 で気をつけないといけないのは、このメソッドをメソッド `setArea2` の前に書くこと。なぜなら `setArea2` の中で呼び出す `drawAxis2` はメソッド 13 ~ 15 によって指定された値を基に枠を描くので、その枠を描いた後に値を変えても枠とグラフが合っていない状態になる。



## 番外編メソッド

### メソッドその 17 drawHideView

これは hideBirdView と全く同じ用途で作られたメソッドだ。使い方も同じだし分割数を増やしても間隔数を増やせば意味が無いところも同じだ。そしてとてつもなく重く描ききるのに結構な時間がかかる。hideBirdView の方は結構早い。hideBirdView さえあればこれは全く必要ない。順序的には drawHideView を作ってから hideBirdView を作ったという感じである。しかしせっかく作ったわけだから載せてみた。

### メソッドその 18 drawAxis2

これは 3 次元グラフを描く時の周りの枠をえがくメソッドである。これは外からは参照できないようになっている。なぜ参照できないかという参照する必要が全く無いからである。これが必要そうなメソッドのなかにはすでにこのメソッドが呼ばれているのである。

メソッド内で drawAxis2 を呼んでいるメソッド。setArea2, drawBirdView, fillBirdView, hideBirdView

## 4.4 MitsuiWorld のアルゴリズム

まず MitsuiWorld を作るにあたって最も参考にした参考書を紹介します。

Java による図形処理入門：山本芳人著 工学図書株式会社

これは MitsuiWorld を作るにあたって大変参考になった。「最も」というよりもむしろ、MitsuiWorld を作るのにはこれしか参考にしていない。逆にいうならば、この本の全てを理解し、この本を数値解析をするのに都合良く変換し、圧縮したのが MitsuiWorld である。MitsuiWorld を読みたい人は参考書にこれをお勧めする。そしてこれからこの章書くアルゴリズムもほとんどこの本からの引用である。だからこのセクションを飛ばしてもらっても一向にかまわない。むしろそれの方が分かり易いかもしれない。しかし、その本には載ってないアルゴリズムも MitsuiWorld には存在する。一度目を通しておくのも損は無いと思う。

### MitsuiWorld を作ろうとした理由。

世の中は幸か不幸か私たち数学科みたいに数学を意識しているものはあまりない。そして Java もその例外では無かったのである。Java の座標の表示の仕方は左上角が ( 0 , 0 ) であって、右下に行くにしたがって座標の値が大きくなる。正確に言うと右に行けば行くほど x の座標があがり、下に行けば行くほど y 座標が上がるのである。例えばこの状態で  $y = x^2$  のグラフを描きなさいを言われたらあなたはどうしますか？まず軸を書くのにも困るのではないだろうか？原点 ( 0 , 0 ) は中心にあるという先入観を持っている人もいることだろう。そこで数学科の考え方で図を描くには Java 座標から数学座標への座標変換を行う必要があるのだ。これを自動的にしてくれるファイルは、C 言語でいうなら代表的なものに GLSC がある。Java では…。いったい何があるというのだ。世界中を探せば一つぐらいはそのようなファイルがあるかもしれない。しかし何といっても探すのがめんどくさい。そして万が一見つかったとしてもそれを自分のやりたいように出来るかどうか分からないし、使いこなせるかもわからない。この辺のリスクを考えると自分で使いやすい物を作った方がいいかなと思い Java 座標から数学座標への変換をするファイルを自分で作ったのである。

#### 4.4.1 2次元の座標変換

この座標変換は決して難しくは無いですね。別に何も参考にしていません。座標変換の方程式は

$$jx = (x - xmin) \frac{w}{xmax - xmin}$$

$$jy = (ymax - y) \frac{h}{ymax - ymin}$$

ただし  $x$  : 数学座標の  $x$ ,  $y$  : 数学座標の  $y$

$jx$  : Java 座標の  $x$ ,  $jy$  : Java 座標の  $y$

$w$  : 画面の横幅のピクセル数,  $h$  : 画面の縦幅のピクセル数

$xmin$  : 数学座標の  $x$  の最小値,  $xmax$  : 数学座標の  $x$  の最大値

$ymin$  : 数学座標の  $y$  の最小値,  $ymax$  : 数学座標の  $y$  の最大値

`move(x1, y1)` で  $(x_1, y_1)$  の値を記憶して `draw(x2, y2)` で  $(x_1, y_1)$  から  $(x_2, y_2)$  まで線をひき  $(x_2, y_2)$  を記憶します。つぎの `draw(x3, y3)` をしたときに  $(x_2, y_2)$  から  $(x_3, y_3)$  まで線をひきます。set() で xmin, xmax, ymin, ymax に値を与えます。setGraphics() で w と h の値を与えます。

#### 4.4.2 3次元の座標の動き

3次元空間に図形を描く場合は、x軸、y軸、z軸を使う。コンピューターの画面は2次元なのでx軸、y軸<sup>1</sup>の回りに回転することによって、図形を立体的に表示することが出来る。

x、y、z軸の回りに図形を回転する方法を解説する。回転角はx、y、z軸のプラスの方向に向かって時計回りを正の角とする。

点  $(x, y, z)$  を x 軸回りに角  $\alpha$ 、y 軸の回りに角  $\beta$  だけ回転した点の座標を  $(x_1, y_1, z_1)$  とする。

1 . y 軸の回りに角  $\beta$  だけ回転した場合を考える。y 軸の回りに角  $\beta$  だけ回転した点の座標を  $(x_0, y_0, z_0)$  とする。原点を中心に y 軸回りに角  $\beta$  だけ回転する移動は、行列を使うと次のようになる。

$$\begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} = \begin{pmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$x_0 = x \cos \beta + z \sin \beta$$

$$y_0 = y$$

$$z_0 = -x \sin \beta + z \cos \beta$$

2 . 点  $(x_0, y_0, z_0)$  を x 軸の回りに角  $\alpha$  だけ回転した点の座標を  $(x_1, y_1, z_1)$  とすると次のようになる。

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & \sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix}$$

$$x_1 = x_0$$

$$y_1 = y_0 \cos \alpha - z_0 \sin \alpha$$

$$z_1 = y_0 \sin \alpha + z_0 \cos \alpha$$

3 . 2つの式をまとめる。

$$x_1 = x \cos \beta + z \sin \beta \quad \cdots x_1 \text{ の位置}$$

$$z_0 = -x \sin \beta + z \cos \beta$$

$$y_1 = y \cos \alpha - z_0 \sin \alpha \quad \cdots y_1 \text{ の位置}$$

$$z_1 = y \sin \alpha + z_0 \cos \alpha \quad \cdots z_1 \text{ の位置}$$

これらの式が3次元の座標変換の基礎になります。

<sup>1</sup>このレポートでは数学で言う3次元の座標とは異なり z 軸を手前にして説明します。

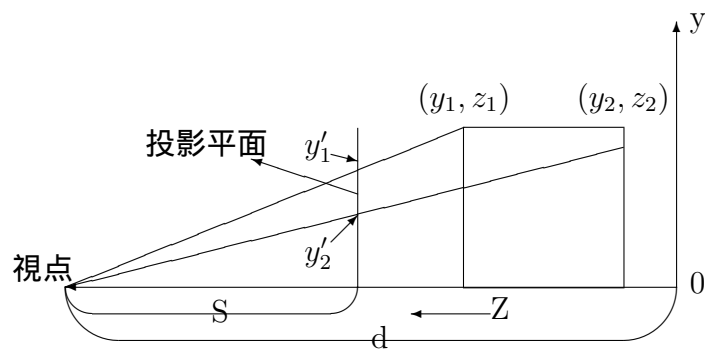
(詳しくは参考書 p 197 ~ )

次にもう一つのアルゴリズムの透視投影を説明します。上の式だけでも 3D のようにすることは出来ます。しかし上の式と透視投影の 2 つの式を合わせることによって遠近感のある 3D を表現します。

#### 4.4.3 透視投影

透視投影は、図形に遠近感をつけるために、空間上の一点を視点とし、視点と原点を結んだ直線上に、この直線と垂直な面 (投影平面) に写った図形を作成する方法である。

この投影法では、遠くのは小さく、近くのは大きく表示されるので遠近感を出すことが出来る。



上の図のように投影平面を設定する。視点から投影平面までの長さを  $s$ 、視点から原点までの長さを  $d$  とする。

三角形の比の関係から次のようになる

$$\begin{aligned} y'_1 : y_1 &= s : (d - z_1) \\ y'_1 &= \frac{s}{d - z_1} y_1 \end{aligned}$$

$$\begin{aligned} y'_2 : y_2 &= s : (d - z_2) \\ y'_2 &= \frac{s}{d - z_2} y_2 \end{aligned}$$

$x$  の値も同様に計算する。

(詳しくは参考書 p 209 ~ )

#### 4.4.4 3次元の座標変換

3次元の座標の動きから求めた式

$$x_1 = x \cos \beta + z \sin \beta \quad \cdots x_1 \text{ の位置}$$

$$\begin{aligned}
z_0 &= -x \sin \beta + z \cos \beta \\
y_1 &= y \cos \alpha - z_0 \sin \alpha \quad \cdots y_1 \text{ の位置} \\
z_1 &= y \sin \alpha + z_0 \cos \alpha \quad \cdots z_1 \text{ の位置}
\end{aligned}$$

と透視投影から求めた式

$$\begin{aligned}
x'_1 &= \frac{s}{d - z_1} x_1 \\
y'_1 &= \frac{s}{d - z_1} y_1
\end{aligned}$$

とで3次元を描くことが出来るが、まだ気おつけて欲しい。この式であたえる  $x, y, z$  というのは  $x_1, y_1, z_1$  と同じ領域の座標なのである。すなわち今考えている Java で言うなら求めたい Java の座標 (ピクセル座標) を得るのに Java の座標 (ピクセル座標) を与えるのである。そこで数学座標でも考えられるようにするために数学座標からピクセル座標への座標変換が必要なのである。

$$\begin{aligned}
jx &= (x - xmin) \frac{lx}{xmax - xmin} \\
jy &= (z - zmin) \frac{lz}{zmax - zmin} \\
jz &= (ymax - y) \frac{ly}{ymax - ymin}
\end{aligned}$$

ただし  $x$ : 数学座標の  $x$ ,  $y$ : 数学座標の  $y$ ,  $z$ : 数学座標の  $z$   
 $jx$ : Java 座標の  $x$ ,  $jy$ : Java 座標の  $y$ ,  $jz$ : Java 座標の  $z$   
 $lx$ :  $x$  軸長さ,  $ly$ :  $y$  軸長さ,  $lz$ :  $z$  軸長さ  
 $xmin$ : 数学座標の  $x$  の最小値,  $xmax$ : 数学座標の  $x$  の最大値  
 $ymin$ : 数学座標の  $y$  の最小値,  $ymax$ : 数学座標の  $y$  の最大値  
 $zmin$ : 数学座標の  $z$  の最小値,  $zmax$ : 数学座標の  $z$  の最大値

ただしここで気おつけて欲しいことがある。前にも述べたと思うが3次元において今まで述べてきたアルゴリズムは全て数学の座標とは違い  $z$  軸が手前向きの方である。このまま使うにはかなり使いにくい。そこで座標変換をする部分で  $y$  の値と  $z$  の値を交換している。そうすることによって自分たちがこの関数を使う時には普通どうり数学座標で考えられるのである。

角  $\alpha, \beta$  の値を変えるのに `changeAngle` をつかい、 $lx, ly, lz$  の値を変えるのに `changeSize` を使い、 $s, d$  の値を変えるのに `changeReflect` をつかい、原点の位置を変えるのに `changePosition` を使うのである。

この  $x'_1, y'_1$  が3次元を2次元にした座標である。あと線を描くのは2次元の時と全く同じである。

#### 4.4.5 BirdView メソッド

これを作った目的は鳥瞰図を描くのを簡単にするためのものである。基本的のこの BirdView のメソッドは `move2` と `draw2` が主体となっている。だから毎回プログラムを書くたびに定義することも出来る。どうせ毎回そのメソッドを使うならそれを新たなメソッドとして定義することによって簡単化し、プログラム自体の行数も減らし、見通しも良くなることだろう。



## drawBirdView

これははっきり言って何も難しくない。move2 と draw2 を使って毎回記述するのはめんどくさいところから作ったメソッドであって move2 と draw2 を使いこなせる人ならきっと誰にでも作れただろう。

この中でしている事は例2 でしている事と全く同じである。ただそれをどんな範囲でも、またどんな分割数でも出来るようにしただけである。

## fillBirdView

アルゴリズムとしては結構簡単だとは思う。最小の格子点の四角形を作りそれを塗りつぶす。全ての格子点の四角形にこれと同じことをする。そして最後に drawBirdView で線をひく。最小の格子点とは  $U_{i,j}^k, U_{i,j+1}^k, U_{i+1,j}^k, U_{i+1,j+1}^k$  でくる四角形のことである。これを  $(0 \leq i \leq Nx-1, 0 \leq j \leq Ny-1)$  の範囲でするわけである。最小の格子点を塗りつぶすのに fillPolygon を使っている。これは与えた点を結ぶ多角形を作ってくれる関数で、いびつな形の多角形を作成するにはもってこいのメソッドである。

## hideBirdView

このアルゴリズムも簡単といえば簡単だが、気づくのにはかなりの時間がかかった。fillBirdView とほとんど同じだが唯一違うのは fillBirdView では全てを塗りつぶしてから drawBirdView で線を描いていたのだが、hideBirdView では最小の四角形を塗りつぶした後にすぐその四角形の輪郭を黒くしているのである。そして一番奥から描き始めることによって奥の背景を消すといったチープな隠面消去法を用いている。この隠面消去法はだぶん数値解析しか使えないだろうと思う。一律で奥のほうから描き始める（描くことが出来る）3Dなんてのはあまり無いような気がする。

しかし数値解析にあたっては大いなる力を発揮する。一目見てわかると思うのだがなんといっても計算量が fillBirdView よりも少ないのである。大げさに言えば fillBirdView を呼ばない分だけ早くなっているのである。

しかしこんないいこと尽くしに見える hideBirdView でも fillBirdView にはかなわない部分が1つある。それは滑らかさである。2つの式を眺めてもらえば気づくかもしれないが最小の四角形の取り方が微妙に違うのである。

違いを例で説明すると  $N_x, N_y$  の分割数を40にした時、

$$\begin{aligned} &fillBirdView(u, N_x, N_y, 1) \\ &hideBirdView(u, N_x, N_y, 1) \end{aligned}$$

は両方とも滑らかさは同じである。ただし、ここで  $u$  は任意の2次元配列とする。

しかし  $N_x, N_y$  の分割数を80にした時、

$$\begin{aligned} &fillBirdView(u, N_x, N_y, 2) \\ &hideBirdView(u, N_x, N_y, 2) \end{aligned}$$

では fillBirdView の方がきれいに描けるのである。ただし、ここで  $u$  は任意の2次元配列とする。理由は簡単である。fillBirdView の方は塗りつぶすのと線をひくのを別々に行っている

が、hideBirdView では並行に行っている。すなわち fillBirdView の方は線で囲まれた部分をさらに分割して最小の四角形を作ることが出来るが、hideBirdView ではそうはいかない。線で囲まれた部分が最小の hideBirdView 四角形なのだ。すなわち hideBirdView は上の例でいうなら分割数が 40 の時と 80 の時とで両方とも全く同じ滑らかさなのだ。

それではなぜそんな引数のわたし方をするのか疑問に思うことだろう。それは BirdView 系の使い方が全部一緒ならこれを使う人が混乱せずに済むと思ったからだ。ユーザーの立場からすれば drawBirdView と hideBirdView の使い方が同じで hideBirdView だけが違っているよりも、3 つとも使い方が同じ方が絶対わかりやすいと思ったからだ。ということでこの辺で BirdView の話を終わりにしたいと思う。

#### 4.4.6 Zバッファによる陰面消去法

一番最初に述べておくがこの方法ほど数値解析に向いてない陰面消去法は無いんじゃないかと思うくらい使えない。読みたくない人はほんとに読まなくていいセクションである。なぜならこの方法を用いて作ったメソッドが drawHideView であり、これのかわりをするのが hideBirdView である。さらに hideBirdView の方が簡単で高速に動くからである。ではなぜあるかという hideBirdView を思いつく前にこの方法を作ったからである。さらにこういうメソッドに限って MitsuiWorld のなかで一番苦労して作り上げたメソッドだったりする。さらに陰面消去法も hideBirdView よりも高度なアルゴリズムである。そんな理由でこのまま捨てるのにはあまりにもむなしすぎるので紹介しておく。

Z バッファによる陰面消去法もアルゴリズムは簡単である。要は全てのピクセルに対して z 座標（手前方向）を持ち、そのピクセルで z 座標が一番大きなものが生き残るといった方法である。聞いただけでもわかるとうり全てのピクセルを何回も比較するのである。ほんとに気が遠くなるぐらいに遅い。遅い。遅すぎる。とてもじゃないけどアニメーションには使えなかった。

プログラムが長いように見えるが 1 ピクセル 1 ピクセル地道に場合わけをしてひたすら比較しているのである。比較の仕方であるが ZB という配列がありこれが全てのピクセルの z 座標を管理するシステムになっている。そして BU という配列に更新ごとを書き込んで ZB と比較するのである。そして ZB より BU の方が z 座標が大きいピクセルがあれば ZB に書き込む。これを永遠と繰り返すのである。

詳しくは参考書の p 229 ~ を参照してください。

## 4.5 MitsuiWorld program

```
/** 数学のグラフを描くのに便利なパッケージを作ってみました。
 *   これを使うと Java の座標を考えずに
 *   数学の座標だけを考慮してグラフを描く事ができます。
 *
 *   名前は気にしないで下さい。自己満足の世界です。
 *
 *   関数の数には満足のいく物がありますが、等高線を描くメソッドを
 *   作れなかったことを大変悔やみます。もう少し考える時間があれば
 *   何とかあったかもしれないのがつらいです。
 *
 *
 *   メソッド一覧 現在 16 個   private なメソッドは入っていません。
 *   共通のメソッド
 *   setGraphics, setColor
 *   二次元グラフのメソッド
 *   set, move, draw
 *   三次元グラフのメソッド
 *   set2, move2, draw2, changeAng, changePosi, changeSize,
 *   changeReflect, drawBirdView, fillBirdView, hideBirdView, drawHideView
 *
 *   このプログラムでは MitsuiWorld をパッケージとして扱っています。
 *   もしこのままパッケージとして使いたいなら Mitsui というディレクトリを作り
 *   そのなかにこの MitsuiWorld のファイルをいれ、そこでこのファイルを
 *   コンパイルします。
 *
 *   使い方は MitsuiWorld を使うプログラムと同じディレクトリ内に上で作った
 *   Mitsui ディレクトリを置きます。
 *   例えば c:/usr/home% というディレクトリに MitsuiWorld を使う Wave.java
 *   があつたとすると Mitsui ディレクトリを home ディレクトリにおき、
 *   Wave.java の初めに必ず
 *
 *                               import Mitsui.*;
 *
 *   と書きます。
 *   これで普通どうりに javac Wave.java とするとコンパイルできます。
 */
```

```
package Mitsui;

import java.awt.*;

public class MitsuiWorld_2{
    //Java の座標
    private int current_x, current_y, next_x, next_y;

    //仮の Z 座標
    private double next_z;

    //[ a , b ] x軸  [ c , d ] y軸  [ e , f ] z 軸
    private double xmin, xmax, ymin, ymax, zmin, zmax;

    //グラフの高さと幅
    private int h, w;

    //カラー
    private Color color;
```

```

//グラフ
private Graphics g,gg;

//角度を変える
private double angX=0,angY=0;

//場所を変える
private int p_x=0,p_y=0;

//三次元グラフの大きさ
private int dlx=0,dly=0,dlz=0;

//透視投影のサイズ
private double s=400,d=500;

//この ZB に全ての点での z 座標が格納される
private float[] [] ZB;

//この BU に ZB と比較する z 座標が格納される
private float[] [] BU;

//色を変更する
private Color[] col={Color.black,Color.white,Color.blue,Color.red,
                    Color.yellow,Color.green,Color.darkGray,Color.gray,
                    Color.lightGray,Color.pink,Color.orange,
                    Color.magenta,Color.cyan};

//Graphics をセットするメソッド。
public void setGraphics(Graphics g){
    this.g = g;
}

//描写画面のサイズのセットをするメソッド。
public void setScreenSize(int width,int height){
    w = width;
    h = height;
}

//色をセットするメソッド。
public void setColor(Color col){
    color = col;
    g.setColor(color);
}

//色をセットするメソッドパート2 番号指定
public void setColor(int c){
    if(c>12)
        c=0;
    color = col[c];
    g.setColor(color);
}

//出力画面を [ a , b ] × [ c , d ] にするメソッド。
public void setArea(double a,double b,double c,double d){
    xmin = a;
    xmax = b;
    ymin = c;
    ymax = d;
}

```

```

}

//ポイントを [X,Y] にもっていくメソッド。
public void move(double x,double y){
    next_x = (int)((x-xmin)*w/(xmax-xmin));
    next_y = (int)((ymax-y)*h/(ymax-ymin));
}

// [X,Y] まで線を引くメソッド。
public void draw(double x,double y){
    current_x = next_x;
    current_y = next_y;
    move(x,y);
    g.drawLine(current_x,current_y,next_x,next_y);
}

//出力画面を [a,b] × [c,d] × [e,f] にするメソッド。(x 軸 × y 軸 × z 軸)
//枠も一緒に作成しています。
public void setArea2(double a,double b,double c,
                    double d,double e,double f){

    xmin = a;
    xmax = b;
    ymin = c;
    ymax = d;
    zmin = e;
    zmax = f;

    drawAxis2();
}

//枠の作成*****private
private void drawAxis2(){
    g.setColor(Color.black);
    move2(xmin,ymin,zmin);draw2(xmax,ymin,zmin);
    draw2(xmax,ymax,zmin);draw2(xmin,ymax,zmin);
    draw2(xmin,ymin,zmin);draw2(xmin,ymin,zmax);
    move2(xmax,ymin,zmin);draw2(xmax,ymin,zmax);
    move2(xmax,ymax,zmin);draw2(xmax,ymax,zmax);
    move2(xmin,ymax,zmin);draw2(xmin,ymax,zmax);
    g.setColor(color);
}

//(x,y,z) にポイントを移動する。
public void move2(double x,double y,double z){
    //数学座標をそのまま Java 座標に変換したものをいれる変数
    int jx,jy,jz;
    //jx,jy,jz から求められる 3次元対応に変換したものを入れる変数
    double x1,y1,z1,z0;

    //枠の長さ
    int lx = w/2 + dlx;
    int ly = w/2 + dly;
    int lz = 3*h/10 + dlz;

    //数学の座標から java の座標に変換
    jx = (int)((x-xmin)*lx/(xmax-xmin));
    jy = (int)((z-zmin)*lz/(zmax-zmin));
    jz = (int)((y-ymin)*ly/(ymax-ymin));
}

```

```

//座標の位置（原点）
int x0 = 5*w/12 + p_x;
int y0 = h/2 - p_y;

//座標の視覚
double RAD = Math.PI/180.0;           //角度をラジアンに直している
double baseAngY = 20.0*RAD;
double baseAngX = -35.0*RAD;

//視覚の角度
double angx = baseAngX + angX*RAD;
double angy = baseAngY + angY*RAD;

//Math.Math. と書くのがめんどくさいしこの方がきれいに書ける
double cosY = Math.cos(angy);
double sinY = Math.sin(angy);
double sinX = Math.sin(angx);
double cosX = Math.cos(angx);

// 3 D対応の変換
x1 = jx*cosX+jz*sinX;
z0 = -jx*sinX+jz*cosX;
y1 = jy*cosY-z0*sinY;
z1 = jy*sinY+z0*cosY;
x1 = s/(d-z1)*x1;           //透視投影
y1 = s/(d-z1)*y1;           //透視投影
next_x=x0+(int)Math rint(x1);
next_y=y0-(int)Math rint(y1);
next_z=z1;
}

//三次元グラフの角度を変えるメソッド
public void changeAngle(double x,double y){
    angX = x;
    angY = y;
}

//三次元グラフの位置を変えるメソッド
public void changePosition(int x,int y){
    p_x = x;
    p_y = y;
}

//三次元グラフの大きさを変える。
public void changeSize(int x,int y,int z){
    dlx=x;
    dly=y;
    dlz=z;
}

//透視投影のサイズを変える。
public void changeReflect(int s,int d){
    this.s=s;
    this.d=d;
}

```

```

//線を描くメソッド
public void draw2(double x,double y,double z){
    current_x=next_x;
    current_y=next_y;
    move2(x,y,z);
    g.drawLine(current_x,current_y,next_x,next_y);
}

//バードビューで見るメソッド
public void drawBirdView(double u[][] ,int nx,int ny,int h){
    double x,y;
    double dx = (xmax-xmin)/nx;
    double dy = (ymax-ymin)/ny;

    for(int i=0;i<=nx;i+=h){
        x = xmin + i * dx;
        move2(x,ymin,u[i][0]);
        for(int j=0;j<=ny;j++){
            y = ymin + j * dy;
            draw2(x,y,u[i][j]);
        }
    }
    for(int j=0;j<=ny;j+=h){
        y = ymin + j * dy;
        move2(xmin,y,u[0][j]);
        for(int i=0;i<=nx;i++){
            x = xmin + i * dx;
            draw2(x,y,u[i][j]);
        }
    }
    drawAxis2();
}

//色つきバードビュー
public void fillBirdView(double u[][] ,int nx,int ny,int h){
    double x,y;
    double dx = (xmax-xmin)/nx;
    double dy = (ymax-ymin)/ny;
    int[] polyx = new int[4];
    int[] polyy = new int[4];

    //最小の四角形を片っ端から求めていく
    for(int i=0;i<nx;i++){
        for(int j=0;j<ny;j++){
            x=xmin+i*dx;
            y=ymin+j*dy;
            move2(x,y,u[i][j]); //点 1
            polyx[0]=next_x;
            polyy[0]=next_y;
            move2(x+dx,y,u[i+1][j]); //点 2
            polyx[1]=next_x;
            polyy[1]=next_y;
            move2(x+dx,y+dy,u[i+1][j+1]); //点 3
            polyx[2]=next_x;
            polyy[2]=next_y;
            move2(x,y+dy,u[i][j+1]); //点 4 これで四角形
            polyx[3]=next_x;

```

```

        polyy[3]=next_y;

        g.fillPolygon(polyx,polyy,4);
    }
}

//線をひく
drawBirdView(u,nx,ny,h);
}

//陰面消去法をした fillBirdView
public void hideBirdView(double u[][],int nx,int ny,int h){
    double x,y;
    double dx = (xmax-xmin)/nx;
    double dy = (ymax-ymin)/ny;
    int[] polyx = new int[4];
    int[] polyy = new int[4];

    drawAxis2();

    for(int i=0;i<nx;i+=h){
        for(int j=0;j<ny;j+=h){
            x=xmin+i*dx;
            y=ymin+j*dy;
            move2(x,y,u[i][j]);
            polyx[0]=next_x;
            polyy[0]=next_y;
            move2(x+h*dx,y,u[i+h][j]);
            polyx[1]=next_x;
            polyy[1]=next_y;
            move2(x+h*dx,y+h*dy,u[i+h][j+h]);
            polyx[2]=next_x;
            polyy[2]=next_y;
            move2(x,y+h*dy,u[i][j+h]);
            polyx[3]=next_x;
            polyy[3]=next_y;

            g.fillPolygon(polyx,polyy,4);          //四角形を塗りつぶす
            g.setColor(Color.black);
            g.drawPolygon(polyx,polyy,4);          //四角形の外枠を描く
            g.setColor(color);
        }
    }

    //一番手前の軸を書き直している
    g.setColor(Color.black);
    move2(xmax,ymax,zmin);
    draw2(xmax,ymax,zmax);
    g.setColor(color);
}

//隠面消去法によるバードビュー
public void drawHideView(double u[][],int nx,int ny,int hh){
    //この ZB に全ての点での z 座標が格納される
    ZB = new float[w+1][h+1];
    //この BU に ZB と比較する z 座標が格納される
    BU = new float[w+1][h+1];

```



```

double x,y;
double dx = (xmax-xmin)/nx;
double dy = (ymax-ymin)/ny;
int[] polyx = new int[4];
int[] polyy = new int[4];
double[] polyz = new double[4];
int[] X = new int[ny+1];
int[] Y = new int[ny+1];
double[] Z = new double[ny+1];

// z バッファの初期化
for(int i=0;i<w;i++){
    for(int j=0;j<h;j++){
        ZB[i][j]=-9999;
    }
}

for(int i=0;i<nx;i+=hh){
    for(int j=0;j<ny;j+=hh){
        x=xmin+i*dx;
        y=ymin+j*dy;
        //線と線の交点の最小の四角形を作る
        move2(x,y,u[i][j]);
        polyx[0]=next_x;
        polyy[0]=next_y;
        polyz[0]=next_z;
        move2(x+hh*dx,y,u[i+hh][j]);
        polyx[1]=next_x;
        polyy[1]=next_y;
        polyz[1]=next_z;
        move2(x+hh*dx,y+hh*dy,u[i+hh][j+hh]);
        polyx[2]=next_x;
        polyy[2]=next_y;
        polyz[2]=next_z;
        move2(x,y+hh*dy,u[i][j+hh]);
        polyx[3]=next_x;
        polyy[3]=next_y;
        polyz[3]=next_z;

        writeZB(polyx,polyy,polyz);
    }
}
//一番手前の軸を書き直している
g.setColor(Color.black);
move2(xmax,ymax,zmin);
draw2(xmax,ymax,zmax);
g.setColor(color);
}

//平面の各ドットごとに描く *****private
private void writeZB(int[] X,int[] Y,double[] Z){
    //x,y,z の初期値と増分
    double xp,xd,yp,yd,zp,zd;
    int XMAX,XMIN,YMAX,YMIN;
    int len = X.length;

```

```

//BUの初期化
for(int i=0;i<w;i++){
    for(int j=0;j<h;j++){
        BU[i][j]=-9999;
    }
}

//四角形（多角形）の中での最大のX,Yと最小のX,Yを決める
XMAX=X[0];XMIN=X[0];YMAX=Y[0];YMIN=Y[0];
for(int i=0;i<len;i++){
    if(X[i]>XMAX) XMAX=X[i];
    if(X[i]<XMIN) XMIN=X[i];
    if(Y[i]>YMAX) YMAX=Y[i];
    if(Y[i]<YMIN) YMIN=Y[i];
}

//この部分で四角形（多角形）の各边上でのz座標が記録される。
for(int k=0;k<len;k++){
    int k1=k;
    int k2=(k+1)%len; //k2=1,2,3,0
    double m=(double)(Y[k2]-Y[k1])/(X[k2]-X[k1]); //傾き

    if(Math.abs(m) <= 1.0){ //傾きが1以下のときxによるループ
        if(X[k2] < X[k1]){
            k1 = k2; //k1 と k2 の交換
            k2 = k;
        }
        yp=(double)Y[k1]; //yの初期値
        yd=(double)(Y[k2]-Y[k1])/(X[k2]-X[k1]); //yの増分
        zp=(double)Z[k1]; //zの初期値
        zd=(double)(Z[k2]-Z[k1])/(X[k2]-X[k1]); //zの増分
        for(int i=X[k1];i<=X[k2];i++){
            //各ドットのzの値を格納
            BU[i][(int)Math rint(yp)] = (float)zp;
            yp+=yd;
            zp+=zd;
        }
    }else{ //それ以外yによるループ
        if(Y[k2] < Y[k1]){
            k1 = k2; //k1 と k2 の交換
            k2 = k;
        }
        xp=(double)X[k1]; //xの初期値
        xd=(double)(X[k2]-X[k1])/(Y[k2]-Y[k1]); //xの増分
        zp=(double)Z[k1]; //zの初期値
        zd=(double)(Z[k2]-Z[k1])/(Y[k2]-Y[k1]); //zの増分
        for(int i=Y[k1];i<=Y[k2];i++){
            //各ドットのzの値を格納
            BU[(int)Math rint(xp)][i] = (float)zp;
            xp+=xd;
            zp+=zd;
        }
    }
}

//z座標の更新（ライン）

```

```

for(int x=XMIN;x<=XMAX;x++){
    for(int y=YMIN;y<=YMAX;y++){
        if(BU[x][y] > ZB[x][y]){
            ZB[x][y] = BU[x][y];
            g.setColor(Color.black);
            g.drawLine(x,y,x,y);
            g.setColor(color);
        }
    }
}

//この部分で四角形（多角形）の内部のZ座標を決める
for(int y=YMIN+1;y<YMAX;y++){
    nex:for(int xe=XMAX;xe>XMIN;xe--){
        if(BU[xe][y] != -9999){ //終了位置を探す
            for(int xs=XMIN;xs<xe;xs++){
                if(BU[xs][y] != -9999){ //開始位置を探す
                    zp=(double)BU[xs][y];
                    zd=(double)(BU[xe][y]-BU[xs][y])/(xe-xs);
                    for(int x=xs+1;x<xe;x++){
                        zp+=zd;
                        BU[x][y]=(float)zp;
                    }
                    break nex;
                }
            }
        }
    }
}

//Z座標の更新（内部）
for(int x=XMIN;x<=XMAX;x++){
    for(int y=YMIN;y<=YMAX;y++){
        if(BU[x][y] > ZB[x][y]){
            ZB[x][y] = BU[x][y];
            g.drawLine(x,y,x,y);
        }
    }
}
}
}

```

## 付 録 A      3 次元波動方程式のプログラム

```
//<applet code = Wave3d_test width=500 height=500></applet>

import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import Mitsui.*;

public class Wave3d_test extends Applet implements ActionListener,Runnable{
    Button b_start ,b_stop;
    Thread th = null;
    boolean runmove=false;                //thread の状態

    int t;
    int wmax,hmax;

    Graphics bg;
    Image buf;

    MitsuiWorld_2 m;
    double xmin,xmax,ymin,ymax,zmin,zmax;
    int nx,ny,nz;
    double dx,dy,dz;

    int n=50;
    double[] [] [] u1 = new double[n+1] [n+1] [n+1];
    double[] [] [] u2 = new double[n+1] [n+1] [n+1];
    double[] [] [] u3 = new double[n+1] [n+1] [n+1];

    public void init(){
        b_start = new Button("start");
        b_start.addActionListener(this);
        add(b_start);

        b_stop = new Button("stop");
        b_stop.addActionListener(this);
        add(b_stop);

        wmax = getSize().width;
        hmax = getSize().height;
        buf = createImage(wmax,hmax);
        bg = buf.getGraphics();

        xmin = -2.0;
        xmax = 2.5;
        ymin = -2.0;
        ymax = 2.5;
        zmin = -1.0;
        zmax = 1.0;
    }
}
```

```

        m = new MitsuiWorld_2();
        m.setGraphics(bg);
        m.setScreenSize(wmax,hmax);
        m.setArea2(xmin,xmax,ymin,ymax,zmin,zmax);

        nx = 40;
        ny = 40;
        nz = 40;
        dx = (xmax-xmin)/nx;
        dy = (ymax-ymin)/ny;
        dz = (zmax-zmin)/nz;
    }

    public void start(){
        if(th==null){
            th = new Thread(this);
            th.start();
        }
    }

    public void run(){
        while(true){
            while(!runmove){
            }
            t++;
            double[] [] [] u = f(t);
            bg.clearRect(0,0,wmax,hmax);
            m.hideBirdView(u[nz/2],nx,ny,1);
            repaint();
            try{
                Thread.sleep(100);          //100 ミリ秒ストップ
            }catch(InterruptedException e){}
        }
    }

    public void stop(){
        if(th!=null){
            th=null;
        }
    }

    public void actionPerformed(ActionEvent e){
        if(e.getSource()==b_start){
            runmove=true;          //再びグラフを描き始める
        }

        //stop ボタンを押したときのアクション
        if(e.getSource()==b_stop){
            if(!runmove){          //一時停止のとき
                t=0;
                repaint();
            }
            else{                  //これで一時停止
                runmove=false;
            }
        }
    }
}

```

```

public void paint(Graphics g){
    if(t==0){
        double[] [] [] u = f(t);
        m.setColor(7);
        bg.clearRect(0,0,wmax,hmax);
        m.hideBirdView(u[nz/2],nx,ny,1);
    }
    g.drawImage(buf,0,0,this);
}

public double[] [] [] f(int t){
    double tau = 0.01;

    double lambdax = tau/dx;
    double lambday = tau/dy;
    double lambdaz = tau/dz;

    double lambdax2 = lambdax * lambdax;
    double lambday2 = lambday * lambday;
    double lambdaz2 = lambdaz * lambdaz;

    if(t==0){
        for(int k=0;k<=nz;k++){
            for(int i=0;i<=nx;i++){
                for(int j=0;j<=ny;j++){
                    u1[k][i][j] = phi(xmin+i*dx,ymin+j*dy,zmin+k*dz);
                }
            }
        }
        return u1;
    }

    if(t==1){
        for(int k=1;k<=nz;k++){
            for(int i=1;i<=nx;i++){
                for(int j=1;j<=ny;j++){
                    u2[k][i][j]=(1.0-lambdax2-lambday2-lambdaz2)*u1[k][i][j]+
                        0.5 * lambdax2 * (u1[k][i-1][j]+u1[k][i+1][j]) +
                        0.5 * lambday2 * (u1[k][i][j-1]+u1[k][i][j+1]) +
                        0.5 * lambdaz2 * (u1[k-1][i][j]+u1[k+1][i][j]) +
                        tau * psi(xmin+i*dx,ymin+j*dy,zmin+k*dz);
                }
            }
        }

        //境界条件
        for(int i=0;i<=nx;i++){
            for(int j=0;j<=ny;j++){
                u2[0][i][j] = u2[1][i][j];
                u2[nz][i][j]= u2[nz-1][i][j];
            }
        }
        for(int j=0;j<=ny;j++){
            for(int k=0;k<=nz;k++){
                u2[k][0][j] = u2[k][1][j];
                u2[k][nx][j]= u2[k][nx-1][j];
            }
        }
        for(int k=0;k<=nz;k++){
            for(int i=0;i<=nx;i++){
                u2[k][i][0] = u2[k][i][1];
                u2[k][i][ny]= u2[k][i][ny-1];
            }
        }
    }
}

```

```

        return u2;
    }

    else{
        for(int k=1;k<nz;k++){
            for(int i=1;i<nx;i++){
                for(int j=1;j<ny;j++){
                    u3[k][i][j]=2*(1.0-lambdax2-lambday2-lambdaz2)*u2[k][i][j]+
                        lambdax2 * (u2[k][i-1][j]+u2[k][i+1][j]) +
                        lambday2 * (u2[k][i][j-1]+u2[k][i][j+1]) +
                        lambdaz2 * (u2[k-1][i][j]+u2[k+1][i][j]) - u1[k][i][j];

                    //境界条件
                    for(int i=0;i<=nx;i++){
                        for(int j=0;j<=ny;j++){
                            u3[0][i][j] = u3[1][i][j];
                            u3[nz][i][j] = u3[nz-1][i][j];
                        }
                    }
                    for(int j=0;j<=ny;j++){
                        for(int k=0;k<=nz;k++){
                            u3[k][0][j] = u3[k][1][j];
                            u3[k][nx][j] = u3[k][nx-1][j];
                        }
                    }
                    for(int k=0;k<=nz;k++){
                        for(int i=0;i<=nx;i++){
                            u3[k][i][0] = u3[k][i][1];
                            u3[k][i][ny] = u3[k][i][ny-1];
                        }
                    }

                    //配列交換
                    for(int k=0;k<=nz;k++){
                        for(int i=0;i<=nx;i++){
                            for(int j=0;j<=ny;j++){
                                u1[k][i][j] = u2[k][i][j];
                                u2[k][i][j] = u3[k][i][j];
                            }
                        }
                    }
                }
            }
        }
        return u3;
    }
}

public double phi(double x,double y,double z){
    double r;
    r = Math.sqrt(sqr(x-0.5) + sqr(y-0.5) + sqr(z-0.5));
    if (r > 1e-4)
        return h(r)/r;
    else
        return 0.0;
}

public double psi(double x,double y,double z){
    double r;
    r = Math.sqrt(sqr(x-0.5) + sqr(y-0.5) + sqr(z-0.5));
    if (r > 1e-4)

```

```

        return -1.0*dh(r)/r;
    else
        return 0.0;
}

public double sqr(double x){
    return x*x;
}

public double h(double r){
    double a = 0.1, b = 0.2;
    if (r >= a && r <= b)
        return 4 * sqr(r-a) * sqr(r-b) / sqr(sqr(b-a));
    else
        return 0.0;
}

public double dh(double r){
    double a = 0.1, b = 0.2;
    if (r >= a && r <= b)
        return 8.0 * (r-a) * (r-b) * (2*r-a-b) / sqr(sqr(b-a));
    else
        return 0.0;
}
}

```



## 関連図書

- [1] J a v aによる図形処理入門  
山本 芳人著 工学図書株式会社  
お薦め度 5 今回のアルゴリズムのヒントをここから得た。
- [2] J a v a言語プログラミングLESSON 上、下  
結城 浩著 S O F T B A N K  
お薦め度 5 最高。ものすごいわかりやすい参考書。J a v aについてここから学んだことは多い。
- [3] S w i n gによるJ a v a G U Iプログラミング  
大村 忠史著 カットシステム  
お薦め度 3 S w i n gについてここから学んだ。
- [4] 速習J a v a S w i n gプログラミング  
Satyarah Pantham 著 S O F T B A N K  
お薦め度 3 S w i n gについてここから学んだ。
- [5] J a v aを用いた一次元熱方程式の数値計算  
大葉敏文 佐藤晴郎著 2001年度卒業研究  
お薦め度 3 これが無ければ僕の卒研もここまではいかなかったでしょう。
- [6] 波動方程式の数値解析  
塩谷光晴 岩見敬太 浅見明著 2001年度卒業研究  
お薦め度 3 波動方程式については始めにここから学びました。
- [7] 独習J a v a  
ジョセフ・オニール著 S H O E I S H A  
お薦め度 2 詳しく書いてあるのはいいのだが、詳しくすぎて何もわからない人が読むにしては難しいかも。
- [8] 全部無料でつくるはじめてのホームページ  
浅岡 省一著 S H O E I S H A  
お薦め度 1 ホームページの作り方を学んだ。
- [9] 楽々L A T E X  
野寺 隆志著 共立出版株式会社  
お薦め度 1 L A T E Xの書き方をここから学んだ。