微分方程式を解く

オブジェクト指向プログラミング特論

只木進一：工学系研究科

常微分方程式で書かれたモデル

- 現象が常微分方程式で記述される場合は多い
- 例:調和振動子

$$m\frac{\mathrm{d}^2 x}{\mathrm{d}t^2} = -kx$$

- 連立1階微分方程式に変形できる

$$m\frac{\mathrm{d}v}{\mathrm{d}t} = -kx$$

$$\frac{\mathrm{d}x}{\mathrm{d}t} = v$$

- 連立1階微分方程式の解法があればよい

Runge Kutta法

- 独立変数をx、従属変数の組をy_iとする。従属変数はn個とする。

$$\frac{\mathrm{d}y_i}{\mathrm{d}x} = f_i\left(\vec{y}, x\right)$$

- $\vec{y}(x)$ に対して$\vec{y}(x+h)$ を以下の手順で求める

$$\vec{k}_1 = \vec{f}\left(\vec{y}(x), x\right),\ \vec{y}_1 = \vec{y} + \frac{h}{2}\vec{k}_1$$

$$\vec{k}_2 = \vec{f}\left(\vec{y}_1, x + \frac{h}{2}\right), \vec{y}_2 = \vec{y}(x) + \frac{h}{2}\vec{k}_2$$

$$\vec{k}_3 = \vec{f}\left(\vec{y}_2, x + \frac{h}{2}\right), \vec{y}_3 = \vec{y}(x) + h \times \vec{k}_3$$

$$\vec{k}_4 = \vec{f}\left(\vec{y}_3, x + h\right)$$

$$\vec{y}(x+h) = \vec{y}(x) + \frac{h}{6} \times \left(\vec{k}_1 + 2\vec{k}_2 + 2\vec{k}_3 + \vec{k}_4\right)$$

- Runge Kutta法は一般的、つまり微分方程式に依存しない。
- 各系は異なる連立微分方程式で記述される

- システム毎にRunge Kutta法を実装するのはダメ
  - 毎回、誤りを含む可能性がある
  - ライブラリとして再利用すべき

- C/C++では、関数ポインタを使って、RungeKutta法を実行する関数に、連立微分方程式を表す関数を渡すことができる。
  - Javaには、ポインタが無い!

- Javaのラムダ式を活用
  - 微分方程式を表すインターフェイス DifferentialEquation
    - 微分方程式を表すメソッド derivative
  - Runge Kutta法のクラス
    - DifferentialEquation. derivativeを呼び出す

インターフェイスの定義

```
@FunctionalInterface
public interface DifferentialEquation {

    public double[] derivatives(double x, double y[]);

}
```

Runge Kutta法

```
public class RungeKutta {

    /**
     * One step from x to x + h
     * @param x initial value of independent valiable
     * @param y initial values of dependent valiables
     * @param h step
     * @param eq class contains differential equations
     * @return next values of dependent valiables
     */
    public static double[] rk4( double x, double y[], double h,
            DifferentialEquation eq) {
        double yy[] = new double[n];
        ……
        for (int i = 0; i < n; i++) {
            yy[i] = y[i] + h6 * (k1[i] + 2. * k2[i] + 2. * k3[i] + k4[i]);
        }
        return yy;
    }
```

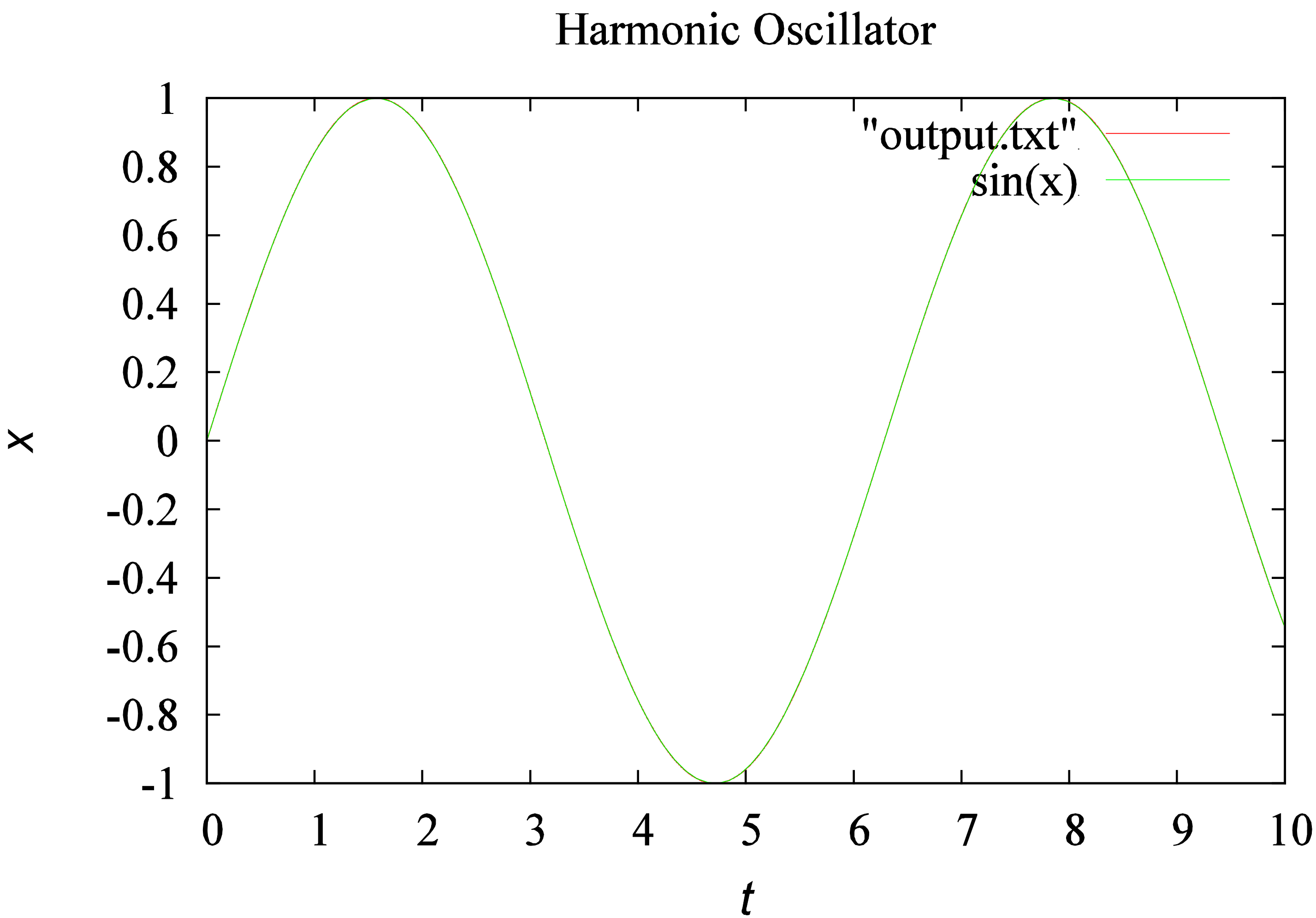
- Runge Kutta法のメソッド内では
  - DifferentialEquation.derivativeの具体を知らなくてよい
  - DeffeerentialEquationインターフェイスを実装したクラスであれば、何でもよい

例：調和振動子：ラムダ式なし

```
DifferentialEquation equation;

 public SimpleCircle(double x, double v, double omega) {
    this.x = x;
    this.v = v;
    //defining equation of motion
    equation = new DifferentialEquation(){
        @Override
        public double[] derivatives(double x, double yy[]){
            double dy[] = new double[2];
            dy[0] = yy[1];
            dy[1] = -omega * omega * yy[0];
            return dy;
        }
    };
}
```

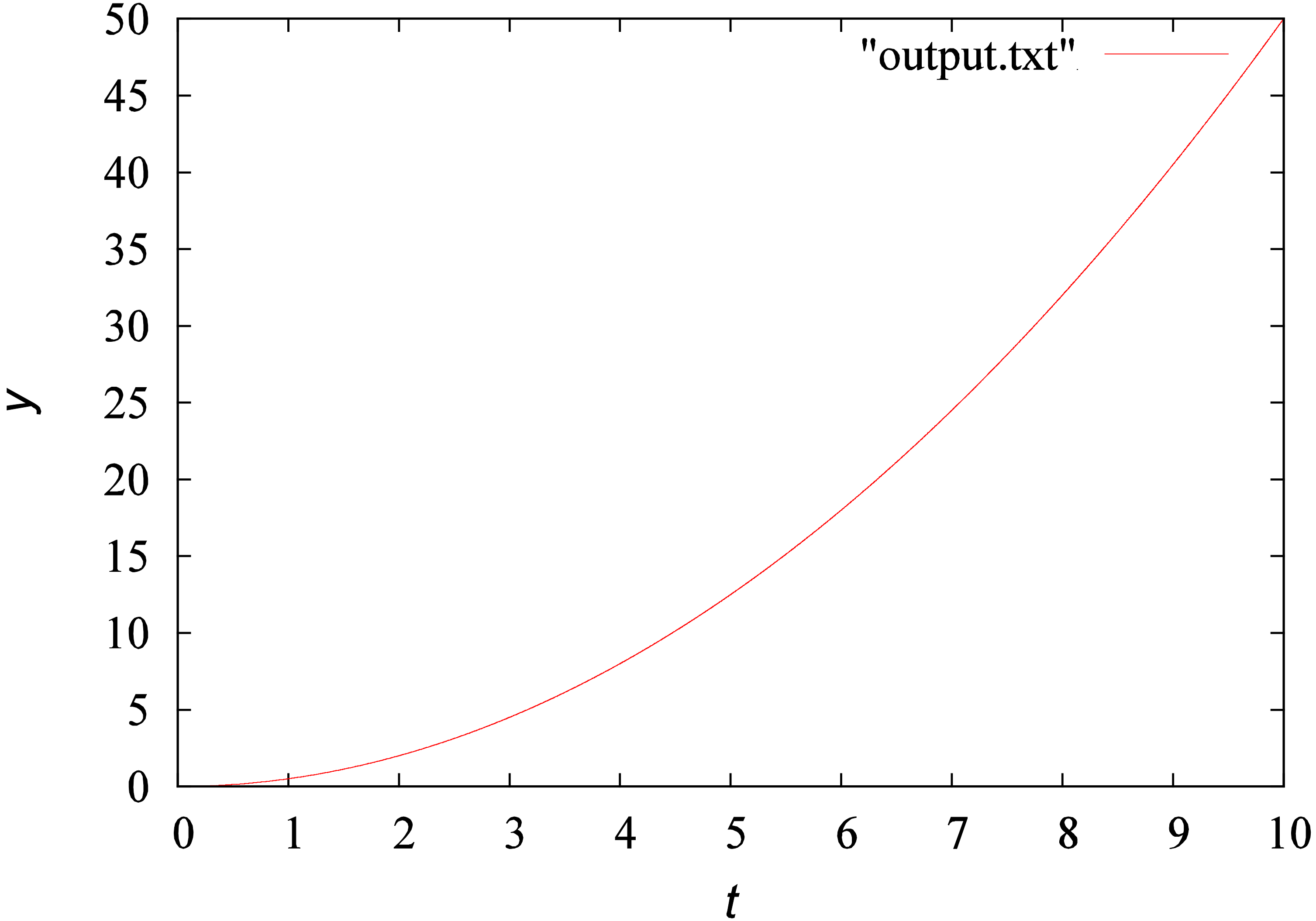
例:調和振動子

```
DifferentialEquation equation;

 public SimpleCircle(double x, double v, double omega) {
     this.x = x;
     this.v = v;
     //defining equation of motion
     equation = (double xx, double[] yy) -> {
         double dy[] = new double[2];
         dy[0] = yy[1];
         dy[1] = -omega * omega * yy[0];
         return dy;
     };
 }
```


Harmonic Oscillator
"output.txt"
sin(x)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1
2
3
4
5
6
7
8
9
10
x
t

例:放物線

```
DifferentialEquation equation;

  public Parabola(double x, double v, double accel) {
    this.x = x;
    this.v = v;
    //defining equation of motion
    equation = (double xx, double[] yy) -> {
      double dy[] = new double[2];
      dy[0] = yy[1];
      dy[1] = accel;
      return dy;
    };
  }
```



50
45
40
35
30
25
20
15
10
5
0
y
"output.txt"
0
1
2
3
4
5
6
7
8
9
10
t