数値計算法・数値解析

常微分方程式：オイラー法

宮崎大学 工学部

第12回

一階常微分方程式の初期値問題

$$
\begin{aligned}
\frac{dy}{dx} = y'(x) &= f(x, y(x)) \\
y(x_0) &= y_0 \qquad \text{：初期条件}
\end{aligned}
$$

が与えられた時，

$x \in [x_0, x_N]$ での $y = y(x)$ の値を求める．

オイラー法の原理：線形近似

$$y_i = y_{i-1} + f(x_{i-1}, y_{i-1})h$$

- $y = y(x)$ は，点 (x_0, y_0) を通る．
- $y = y(x)$ 上の点 (x_0, y_0) での傾きは $f(x_0, y_0)$

点 (x_0, y_0) で $y(x)$ を１次式で近似．

$$y = f(x_0, y_0)(x - x_0) + y_0$$

この近似式で，$y_1 = y(x_1)$ を求める（$x_1 = x_0 + h$）．

$$\begin{aligned} y_1 &= f(x_0, y_0)(x_1 - x_0) + y_0 \\ &= y_0 + f(x_0, y_0)h \end{aligned}$$

常微分方程式の数値解法：オイラー法

- 微分関数 $y' = f(x, y)$
- 区間 $[x_0, x_N]$，分割数 N
- 初期条件 $(x_0, y_0) : y(x_0) = y_0$

が与えられた時，
下式により $(x_i, y_i)(i = 1 \ldots N)$ を順次求める．

$$
\begin{aligned}
y_i &= y_{i-1} + f(x_{i-1}, y_{i-1})h = y(x_i) \\
x_i &= x_0 + ih \\
h &= \frac{x_N - x_0}{N} \quad \text{：刻み幅}
\end{aligned}
$$

例題

$$y' = y, \quad x_0 = 0, y_0 = y(0) = 1, x_N = 1$$

の時，分割数 $N = 4$ として，オイラー法で y(1) の値を求めよ．

オイラー法の原理：テイラー展開

$$\begin{cases} \frac{dy}{dx} = y'(x) & = f(x, y(x)) \\ y(x_0) & = y_0 \end{cases} \quad \text{：初期条件}$$

$y = y(x)$ を $x = x_1$ でテイラー展開すると，

$$\begin{aligned} y(x_1) &= y(x_0 + h) \\ &= y(x_0) + y'(x_0)h + 1/2y''(x_0)h^2 + ... \end{aligned}$$

hの１次の項までで近似すると，

$$\begin{aligned} y(x_1) &= y(x_0) + y'(x_0)h \\ &= y_0 + f(x_0, y_0)h \quad \Rightarrow \text{オイラー法} \end{aligned}$$

オイラー法の誤差

$$
\begin{aligned}
y(x_1) &= y(x_0) + y'(x_0)h + 1/2y''(x_0)h^2 + \dots \\
&\sim y(x_0) + y'(x_0)h
\end{aligned}
$$

オイラー法は，h^2 以上の項を無視．

無視した項の影響がどんどん蓄積されるので，
精度が悪くなる．

【対処法】
h の２次以上の項も考慮した式を考える＝ルンゲクッタ法
次回，紹介します．

オイラー法のプログラム

```
double func1(double x, double y)
{
  return y;
}
int main(int argc, char *argv[])
{ int      n = 10;      // N  分割数
  double   x0 = 1.0;    // x0 区間の先頭
  double   y0 = 2.0;    // y0 初期条件
  double   xn = 4.0;    // xN 区間の末尾

  printf("n:␣%d␣ans:␣y(%g)␣=␣%g\n",
         n, xn, euler(func1, x0, y0, xn, n));

  return 0;
}
```

オイラー法のプログラム

```c
double euler(double (*func)(double, double),
      double x0, double y0, double xn, int n)
{
  double      h = (xn - x0)/n; // 刻み幅
  double      x = x0;
  double      y = y0;

  for(int i = 1; i <= n; i++) {
    y = y + func(x, y) * h;
    x = x0 + i * h;
    printf("%g␣%g\n", x, y);
  }
  return y;
}
```