数値計算法

第六回
gnuplotを使った最小二乗法
微分方程式の解法

若狭　智嗣

粒子物理学講座

リダイレクションについて

- 前回のレポート課題で以下の様に思った者が多いのではないだろうか？
  計算結果（個々のデータの偏差値）が画面では無く、ファイルに出力されれば、
  そのファイルを印刷するだけなので手間がかからない
  （画面の値をレポートに書き写すのは手間）
- UNIXにはリダイレクションという機能があり、以下のことが可能
  - 標準入力（キーボード）　→ ファイル入力に変更
  - 標準出力（画面）　→ ファイル出力に変更
- 例えばファイルの内容を画面に表示するプログラムに cat（14頁）があるが
  - % cat weight.dat　（ファイルweight.datの内容を画面に表示）
  - % cat weight.dat > weight2.dat　（**> がリダイレクションの記号**）
    - ファイルweight.datの内容をファイルweight2.datに書き出す
    - コピー（% cp weight.dat weight2.dat）に等しい
    - 内容が等しい事は diff コマンドで確かめられる
      - % diff weight.dat weight2.dat（同一なので何も表示されない）
    - 書き出すファイルは
      - 無ければ新たに作成される
      - 在る場合は上書きされる（注意！！！）

例題（mean）の場合

- 例題のmeanの場合、標準では

```
% mean ↵
file name :  weight.dat ↵
1:  0.7350 0.2059
2:  0.8860 0.3569
3:  0.7810 0.2519
....
```

と入力ファイル名を聞いてきて、結果（各データと偏差）は画面に出力される

- ここで、リダイレクションを使って結果をファイル(result.dat)に出力させようと

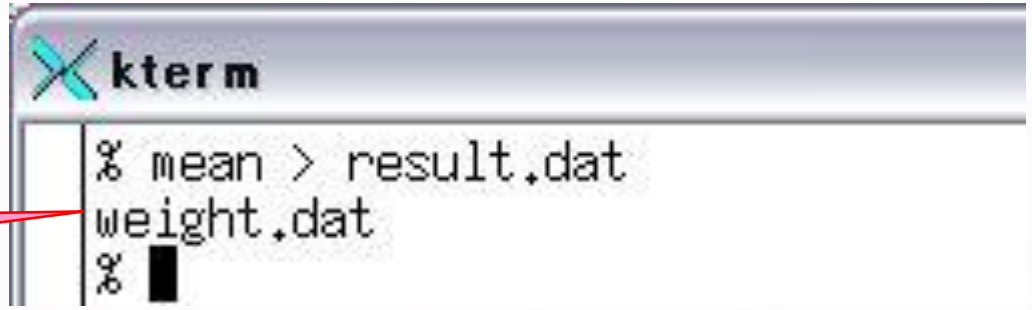
  % mean > result.dat

とすると、プロンプト（%）が返ってこない（入力待ちになる）

  - 実は入力ファイル名の入力を待っている
    - 今まで ”file name :” と画面に表示されていたものが、リダイレクションによりファイル result.dat に書かれている
    - 入力ファイル名 weight.dat を入力すればよい
      （プロンプトが返ってくる）

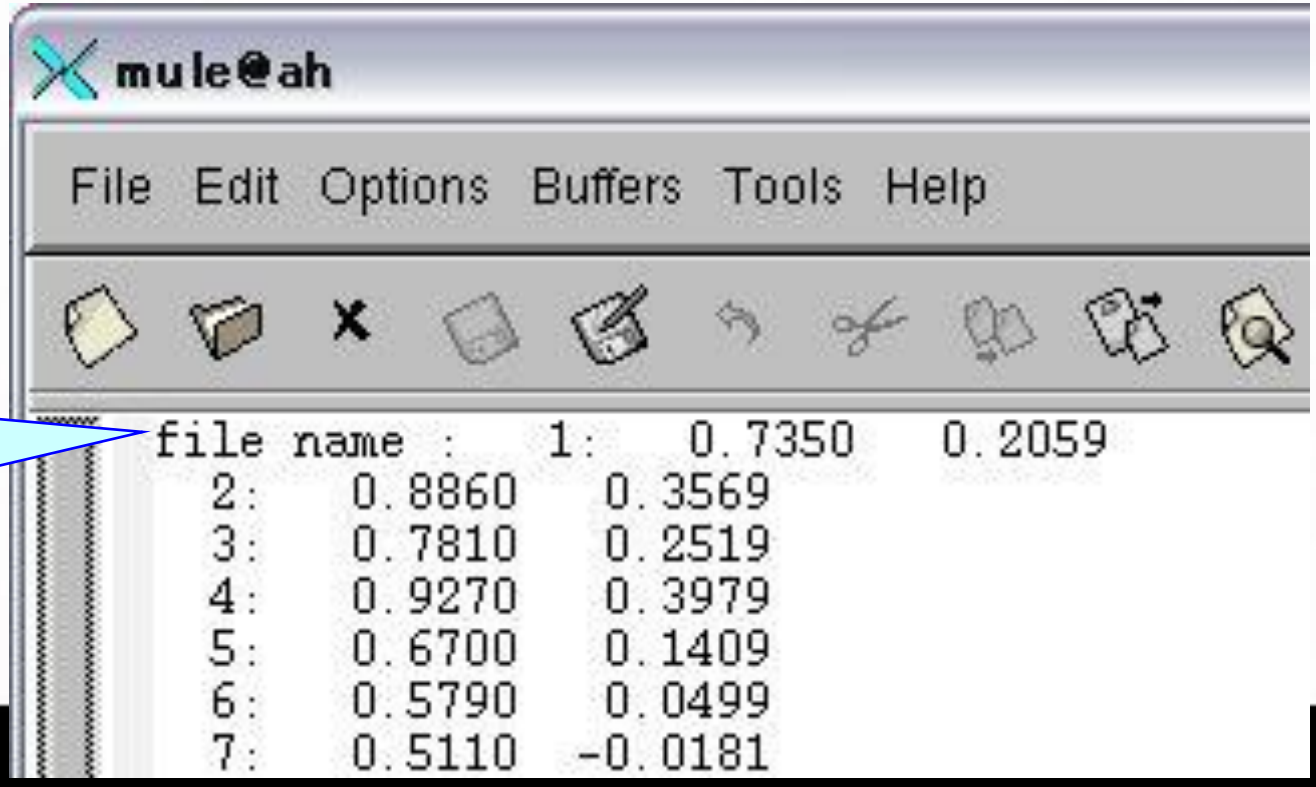
リダイレクションの例

- リダイレクションを使った mean の実行例



ファイル名の入力待ち

- 結果(result.dat)



ファイル名の入力を促すメッセージがファイルにリダイレクトされた

誤差グラフ(測定値に誤差がある場合)

- **データの書式: $(x, y \pm \delta y)$ の場合**
  - 1行に

    x y δy

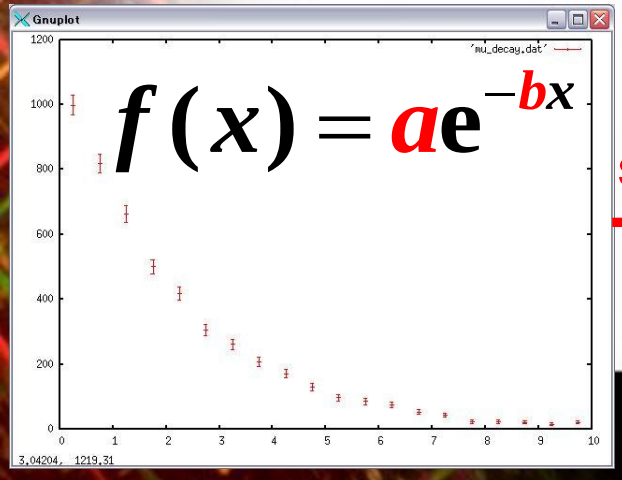
    がスペースで区切られたものを用意
- **書式**
  - gnuplot> plot ‘ファイル名’ with yerrorbars
  - 例
    - gnuplot> plot ‘mu_decay.dat’ with yerrorbars
  - mu_decay.datは以下にある

    /home/teacher/z6wt01in/SAMPLE/mu_decay.dat

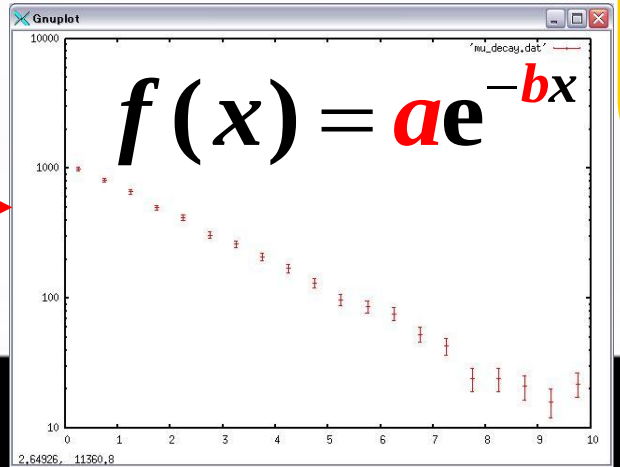
mule@ah

File Edit Options Buffers

```
0.25 999 31.6
0.75 818 28.6
1.25 662 25.7
1.75 501 22.4
2.25 419 20.5
2.75 307 17.5
3.25 262 16.2
3.75 208 14.4
4.25 171 13.1
4.75 131 11.4
5.25  98  9.90
5.75  86  9.27
6.25  76  8.72
6.75  53  7.28
7.25  43  6.56
7.75  24  4.90
8.25  24  4.90
8.75  21  4.58
9.25  16  4.00
9.75  22  4.69
```



set logscale y



Gnuplotによる最小二乗フィット

- **Gnuplotには、非線形関数の最小二乗法が組み込まれている**
  - プログラムを組まなくても手軽に使える
  - 結果をすぐに視覚化できる
- **手順（μ 粒子の寿命測定（決定）を例に）**
  - データファイルを用意
    （例えばx,y,dyの並び）
  - 関数を定義

```
gnuplot> f(x)=a*exp(-b*x)
```

  - 変数の初期値を設定（正しく収束させるために必要）

```
gnuplot> a=1000
gnuplot> b=0.5
```

  - 実行

```
gnuplot> fit f(x) 'mu_decay.dat' using 1:2:3 via a,b
```

データの並びが x,y,δ yである

変数a,bを最適化する

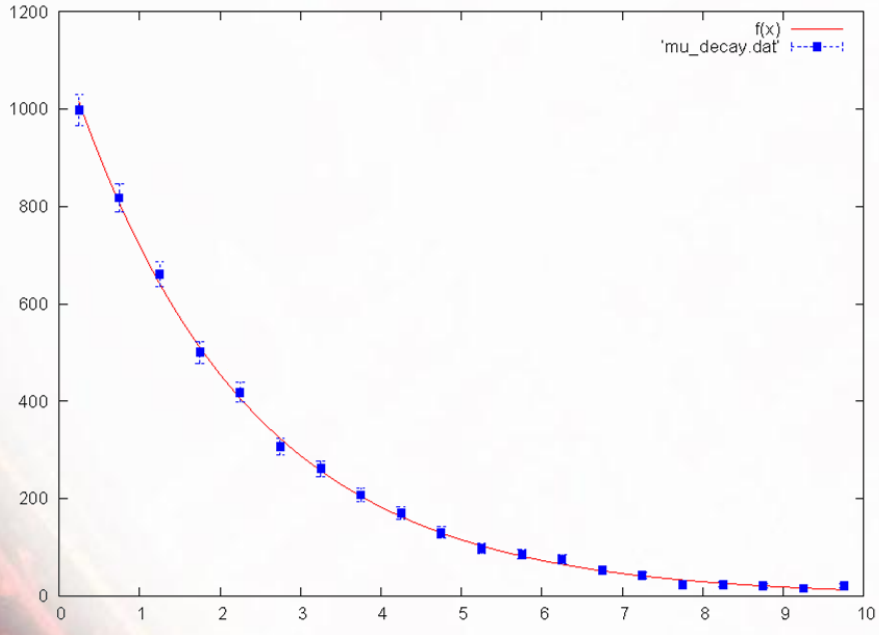
Gnuplotによる最小二乗フィット(続き)

- 手順(続き)
  - 結果

```
Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 1137.76          +/- 19.49        (1.713%)
b               = 0.455834         +/- 0.006014     (1.319%)
```

  - 結果の表示

```
gnuplot> plot f(x), 'mu_decay.dat' with yerrorbars 3 5
```

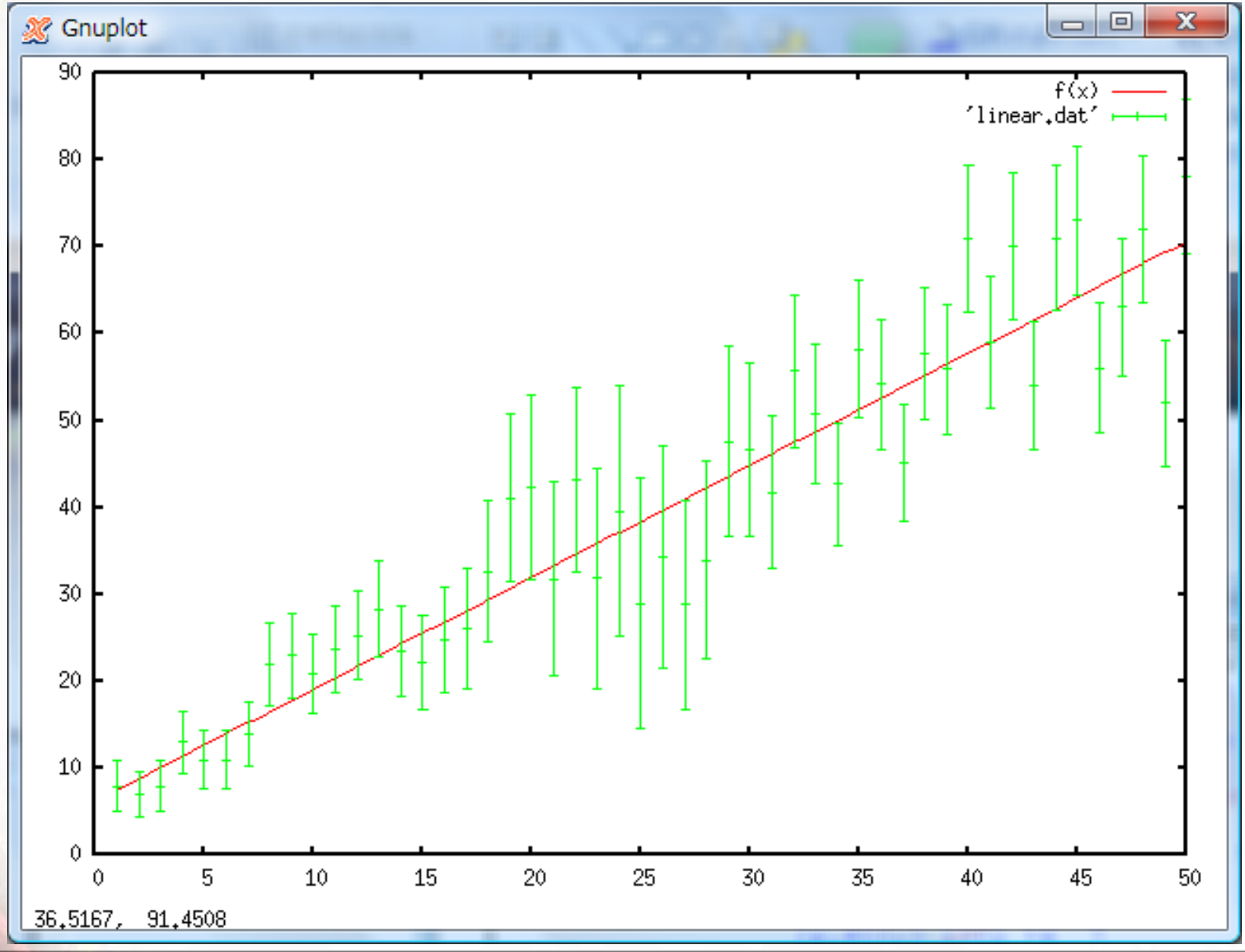


$$\tau = \frac{1}{b} = 2.19 \pm 0.03 \ (\mu \mathrm{s})$$

演習

- xの1次式に従う物理量yとその誤差δyを、いろいろなxについて測定したデータを、
  /home/teacher/z6wt01in/SAMPLE/linear.dat
  におく。
- データの並びは
  x y δy
  である。
- 演習
  - データをgnuplotを用いてグラフにせよ。
  - gnuplotを用いて最小二乗法によるフィッティング（関数の当てはめ）を行い
    y = ax + b
    としたときの係数 a と b を求めよ。
  - gnuplotを用いて、データと回帰直線（最小二乗法から求めた直線）を重ねて描け。

フィッティングの結果


Gnuplot
f(x)
'linear.dat'
90
80
70
60
50
40
30
20
10
0
0
5
10
15
20
25
30
35
40
45
50
36.5167, 91.4508

FORTRANの代表的構文1
do ループ

- do i=1,100
  …
  …
  …
  …
  end do

iが 1 から 100 まで1つずつ増加しながら
100回繰り返される（ループ）
・ループ文の中で、iを参照することは可
j = i　　○
・ループ文の中で、iの値を変える事は不可
i = i + 1　×

- do i=1, 100, 2
  …
  …
  …
  end do

iが 1 から 100 まで2つずつ増加しながら
50回繰り返される（ループ）

FORTRANの代表的構文2
関係式の表現

関係式	FORTRAN での表現
$a = b$	a.eq.b
$a \neq b$	a.ne.b
$a < b$	a.lt.b
$a \leq b$	a.le.b
$a > b$	a.gt.b
$a \geq b$	a.ge.b
$a \wedge b$	a.and.b
$a \vee b$	a.or.b
$\neg a$	.not.a

EQual

Not Equal

Less Than

Less or Equal

Greater Than

Greater or Equal

FORTRANの代表的構文3
do while ループ

- do while (条件式)
  …
  …
  …
  end do

条件式が真である間、繰り返される（ループ）

- i = 1
  do while (i .le. 10)
  …
  …
  i = i + 1
  end do

iを増やさないと、このループは無限に回り続ける（無限ループ!!!）

同じ意味

- do i=1, 10
  …
  …
  end do

FORTRANの代表的構文4
if 文

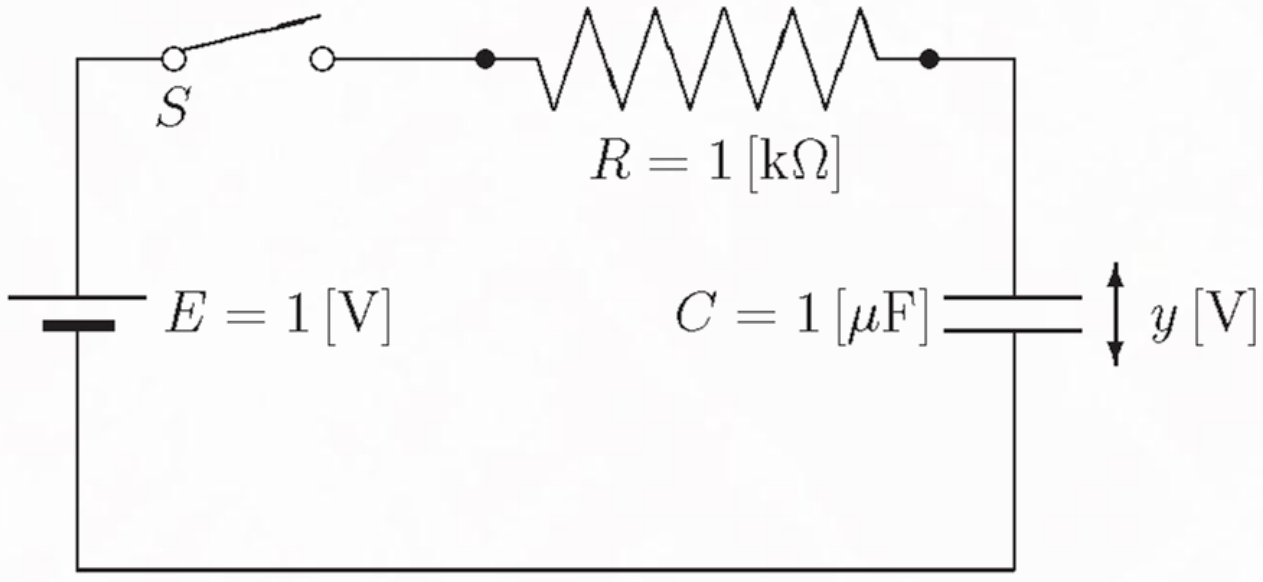
- **条件判断を行う**

```
if (a.gt.b) then                    ! aがbより大きい(.gt.)場合
   statement 1                      ! →statement1を実行
else if (a.eq.b) then               ! aとbが等しい(.eq.)場合
   statement 2                      ! →statement 2を実行
else                                ! 上記何れの条件にも合致しない場合
   statement 3                      ! →statement 3を実行
end if                              ! endifと書いても良い
```

else if(elseifと書いても良い)以降の条件判断は、それ以前の条件判断が真の場合は判断されない事に注意せよ。

微分方程式の数値解法

- **例題**
  - 以下のRC回路を考える。始めコンデンサCに電荷は蓄えられていないとする。時刻 $x=0$ でスイッチSを入れる時、時刻 x でのコンデンサCの両端の電圧 y を求めよ。



微分方程式

- 回路を流れる電流 i は、コンデンサに蓄えられた電荷を Q として、

$$i = \frac{dQ}{dt} = \frac{d(CV)}{dt} = C\frac{dV}{dt}$$

- キルヒホッフの法則から

$$E = V + iR = V + CR\frac{dV}{dt}$$

- $t=x,\ V=y,\ C=1\mu F,\ R=1k\Omega,\ E=1V$より

$$\frac{dy}{dx} = 1 - y \qquad \text{初期条件}: y(0) = 0$$

  - 時定数$\tau=RC$=1 ms であるので、時刻 x は ms で表す（電圧 y はV）

数値微分の考え方

- 数値計算では、変数を離散的に扱う
  - 時刻 x の区間 $[a,b]$ を適当なステップ h で分割
  - $x_n = a + nh\ (n=0,1,2,\ldots)$ における解 y_n を順次求めればよい
    （微分方程式は、xの微小変化(h)に対するyの変化を与えている）

- 順次求める方法
  - Euler法 （オイラー法）
    - 誤差の程度は$O(h^2)$
  - Runge-Kutta法 （ルンゲ-クッタ法）
    - 誤差の程度は$O(h^5)$
    - オイラー法より格段に優れている
  - 精度は h を小さくすれば、原理的には向上する
    - hを小さくすると計算回数は増える
    - 計算精度（桁落ちなど）が問題となる

メインプログラム(rc_circuit)の説明
(/home/teacher/z6wt01in/SAMPLE/rc_circuit.f)

```
      program rc_circuit
      implicit none
c constants:
      real TIME_INIT,TIME_LAST                ! 計算する時間範囲
      parameter(TIME_INIT=0.0,TIME_LAST=4.0)
      real Y_INIT                             ! 電圧の初期値 (V)
      parameter(Y_INIT=0.0)
      real STEP                               ! 時間の刻み (ms)
      parameter(STEP=0.25)
c local:
      real x,y,y_next                         ! 時刻、電圧
c begin:
      y    = Y_INIT                           ! 電圧を初期化
      x    = TIME_INIT                        ! 時間を初期化
      write(*,'(a,F5.2,a,F8.5)')              ! 電圧の初期値を出力
     &     'Time: ',x,' Voltage: ',y
```

x=0 から4 ms まで0.25 ms 間隔で計算

メインプログラム(rc_circuit)の説明(続き)
(/home/teacher/z6wt01in/SAMPLE/rc_circuit.f)

do while文で
1. TIME_LASTまで
2. STEP刻みで

計算する

```
      do while (x.lt.TIME_LAST)
c 時間 x での電圧を y とし、時間が STEP だけ経過した時の
c 電圧 y_next を微分方程式を解いて求める
            call calc_next_step(x,y,STEP,y_next)
            x     = x + STEP                        ! STEP 後の時間
            y     = y_next                          ! STEP 後の電圧
            write(*,'(a,F5.2,a,F8.5)')              ! STEP 後の時間と
     &        'Time: ',x,' Voltage: ',y             ! 電圧を出力
      end do
      stop
      end
```

calc_next_stepとdiff_eqの説明

(/home/teacher/z6wt01in/SAMPLE/diff_eq.f)

- サブルーチン calc_next_step(x,y,STEP,y_next)
  - 離散化された n 番目の値 (x_n,y_n) から $n+1$ 番目の値 (x_{n+1},y_{n+1}) を計算
  - (x_n,y_n) = (x,y), (x_{n+1},y_{n+1}) = (x+STEP,y_next)
  - Euler法なりRunge-Kutta法を用いる(後述)
- 微分方程式の記述

```
      real function diff_eq(x,y)
      implicit none
c constant:
      real MAX_VOLT                          ! 最大電圧（電池電圧） E
      parameter(MAX_VOLT=1.0)
c inputs:
      real x,y                               ! 時刻 x, 電圧 y
c begin:
      diff_eq = MAX_VOLT - y                 ! dy/dx
      return
      end
```

$$\frac{dy}{dx} = E - y$$

微分方程式の解法1
―Euler法―

- 微分の定義

$$\left.\frac{dy}{dx}\right|_{x=x_0} = \lim_{h\to 0}\frac{y(x_n+h)-y(x_n)}{h} \equiv f(x_n, y(x_n))$$

- Euler法（一番簡単な微分方程式の数値解法）

$$y(x_n+h) = y(x_n)+hf(x_n, y(x_n))$$

$$\begin{aligned} y(x_1) &= y(x_0) + hf(x_0, y(x_0)) \\ y(x_2) &= y(x_1) + hf(x_1, y(x_1)) \\ &\vdots \\ y(x_m) &= y(x_{m-1}) + hf(x_{m-1}, y(x_{m-1})) \end{aligned}$$

– 前の点(x_{m-1},y_{m-1})の傾き（微分）をたどって、次の点(x_m,y_m)を求める

Euler法のサンプルプログラムの説明

(/home/teacher/z6wt01in/SAMPLE/euler.f)

```
      subroutine calc_next_step(x,y,step,y_next) ! Euler 法による解法
      implicit none
c inputs:
      real x,y,step                              ! 時刻 x, 電圧 y, 刻み幅 step
c output:
      real y_next                                ! 刻み幅後の電圧
c function:
      real diff_eq                               ! dy/dx
c begin:
      y_next = y + step * diff_eq(x,y)           ! Euler 法
      return
      end
```

$$y(x_n + h) = y(x_n) + h\left.\frac{dy}{dx}\right|_{x=x_n}$$

演習

- サンプルプログラムは
  /home/teacher/z6wt01in/SAMPLE/
  に
  rc_circuit.f diff_eq.f euler.f
  として置いてある

- 演習0：各人コピーし、実行ファイル rc_curcuitを
  % frt -o rc_curcuit rc_circuit.f diff_eq.f euler.f
  により生成し実行せよ
  （コンパイル時に”jwd2008i-i “diff_eq.f”, line 1: この仮引数‘x’は,副プログラム中で使用されていません.”というメッセージが出るが無視して構わない）
- 演習1：厳密解 $y(x)=1-e^{-x}$と比較せよ
- 演習2：刻み幅 STEP を変えて実行し、厳密解との差がどうなるか検討せよ
  （但し、0.0001より小さくしないこと！！！）
- 演習3：計算結果をファイルに出力せよ（リダイレクションを用いよ）
- 演習4：計算結果を gnuplot を用いてグラフにせよ
- 演習5：計算結果と厳密解を gnuplot を用いてグラフにせよ

厳密解との比較の仕方（例）

```
c 時間xでの電圧をyとし、時間がSTEPだけ経過した時の
c 電圧y_nextを微分方程式を解いて求める
      call calc_next_step(x,y,STEP,y_next)
      x     = x + STEP                          ! STEP後の時間
      y     = y_next                            ! STEP後の電圧
      write(*,'(a,F5.2,a,F8.5)')                ! STEP後の時間と
     &   'Time: ',x,' Voltage: ',y               ! 電圧を出力
```

```
c 時間xでの電圧をyとし、時間がSTEPだけ経過した時の
c 電圧y_nextを微分方程式を解いて求める
      call calc_next_step(x,y,STEP,y_next)
      x     = x + STEP                          ! STEP後の時間
      y     = y_next                            ! STEP後の電圧
      write(*,'(a,F5.2,2(a,F8.5))')             ! STEP後の時間と
     &   'Time: ',x,' Voltage: ',y,              ! 電圧、厳密解との差を出力
     &   ' Diff: ',y-(1.0-exp(-x))
```

出力例

```
Time:  0.25 Voltage:  0.25000 Diff:  0.02880
Time:  0.50 Voltage:  0.43750 Diff:  0.04403
Time:  0.75 Voltage:  0.57812 Diff:  0.05049
Time:  1.00 Voltage:  0.68359 Diff:  0.05147
```

結果のSTEP依存性

- **単純な予想**
  - Euler法の誤差はh^2に比例するので、hを1/10にすれば、厳密解との差（誤差）は1/100になる
- **計算誤差を考慮した予想**
  - 計算数はh^{-1}に比例し、数が増えると桁落ちなどの計算誤差が累積する。従って厳密解との差はh^2ではなく、hに比例する
- **結果**
  - STEP=0.1　Time: 4.00 Voltage: 0.98522 Diff: 0.00353
  - STEP=0.01　Time: 4.00 Voltage: 0.98205 Diff: 0.00037
  - STEP=0.001　Time: 4.00 Voltage: 0.98165 Diff: 0.00004
  - STEP=0.0001　Time: 4.00 Voltage: 0.98163 Diff: 0.00004

1/10

1/10

桁落ちの影響が支配的になり精度が向上しない

結果の視覚化(グラフ化)

- **結果をファイルに出力する(リダイレクション)**
  - % rc_circuit > rc_circuit.dat
- **Gnuplotによるグラフ描画**
  - gnuplot> plot 'rc_circuit.dat' using 2:4, 1-exp(-x)

厳密解

2列目をx

4列目をy