平成28年度数学特別講究
Numerical Solution of Stochastic Differential Equations

伊藤克哉

平成 28 年 9 月 27 日

目 次

第1章　常微分方程式の時間離散近似

1.1 Introduction

$$\dot{x} = \frac{dx}{dt} = a(t,x),\ \ x(t_0) = x_0 \tag{1.1.1}$$

というような (決定論的) 微分方程式の初期値問題を解くことは多くの場面において必要となるが, 一般にこの微分方程式の解 $x = x(t;t_0,x_0)$ を明示的に求めることは出来ない. また解が見つかったとしても, 明示的でなかったり数値解析の視点からすると複雑で計算や描画に不向きであることが多い.

ここで広く使われている方法が**時間離散近似** である. 時間離散近似は, 与えられた時間離散化 $t_0 < t_1 < t_2 < \cdots < t_n < \cdots$ に対して, 解 $x(t_1;t_0,x_0), x(t_2;t_0,x_0), \cdots, x(t_n;t_0,x_0), \cdots$ を近似するような, $y_1, y_2, \cdots, y_n, \cdots$ を生成する方程式である. この章ではまずこれらの近似方法の一覧を示し, 次にその離散化誤差, 整合性, 安定性, 収束といった性質を示す.

1.2 時間離散近似方法

(1.1.1) のような初期値問題を解くときに一番シンプルな方法が以下の Euler 法である.

方法 1.2.1 (Euler 法) **Euler 法**は $t_0 < t_1 < t_2 < \cdots < t_n < \cdots$ という間隔 $\Delta_n = t_{n+1} - t_n$ の離散化に対して, 近似解を

$$y_{n+1} = y_n + a(t_n, y_n)\Delta_n,\ \ y_0 = x_0 \tag{1.2.1}$$

によって与える.

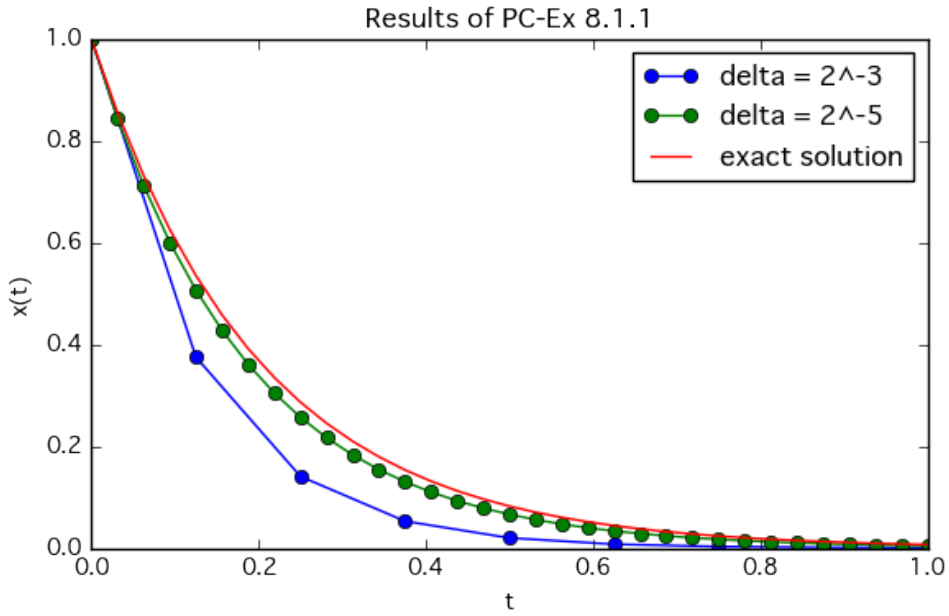
例 1.2.2 *Euler* 法 ($\Delta = 2^{-3}, 2^{-5}$) を次の微分方程式の初期値問題に適応する.

$$\frac{dx}{dt} = -5x, x(0) = 1,\ 0 \le t \le 1$$

In[]:
```
import matplotlib.pyplot as plt
import math
import numpy as np

def euler_method(a,x0,delta):
    t =[delta*x for x in range(int(1/delta)+1)]
    y =[x0]
    for tn in t[:-1]:
        y.append(y[-1]+a(tn,y[-1])*delta)
    return [t,y]
```

In[]:
```
def a811(t,x):
    return -5*x

plt.title("Results of PC-Ex 8.1.1")
plt.xlabel("t")
```

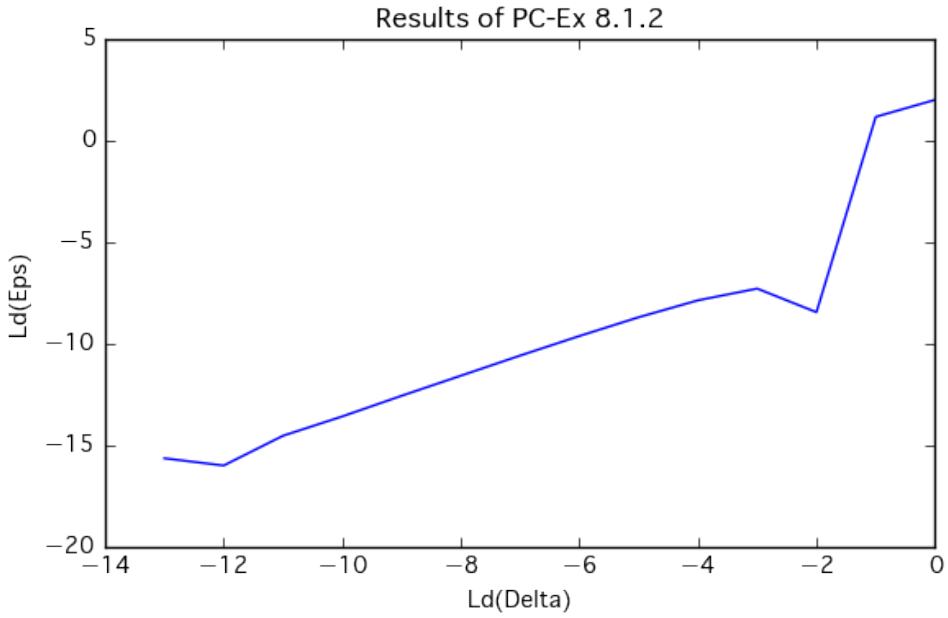
```
plt.ylabel("x(t)")

ans1 = euler_method(a811,1.0,2**(-3))
ans2 = euler_method(a811,1.0,2**(-5))

exact_x = [x*(2**(-5)) for x in range(2**5 + 1)]
exact_y = [math.exp(-5*x) for x in exact_x]

plt.plot(ans1[0],ans1[1],"-o",label="delta = 2^-3")
plt.plot(ans2[0],ans2[1],"-o",label="delta = 2^-5")
plt.plot(exact_x,exact_y ,label = "exact solution")
plt.legend()
plt.show()
plt.close()
```



例 1.2.3 例 1.2.2 と同じ微分方程式の初期値問題に対して,*Euler* 法の $t = 1$ での大域離散化誤差を計算する. 離散化の間隔は $\Delta = 1, 2^{-1}, 2^{-2}, \cdots, 2^{-13}$ とする. 各計算において, 数値を有効数字 5 桁で丸める. 表は,y 軸に $\log_2$ の大域誤差,x 軸に $\log_2 \Delta$ をとってプロットした.
ただし, 大域離散化の定義は,(1.3.1) を参照.

In[]:
```
def roundoff(x,i=5):
    return float(format(x,'.' + str(i) + 'g'))

def euler_method_round(a,x0,delta):
    t =[roundoff(delta*x) for x in range(int(1/delta)+1)]
    y =[x0]
    for tn in t[:-1]:
        y.append(roundoff(y[-1]+a(tn,y[-1])*delta))
    return [t,y]


plt.title("Results of PC-Ex 8.1.2")
plt.xlabel("Ld(Delta)")
plt.ylabel("Ld(Eps)")
x=[i for i in range(0,-14,-1)]

from math import fabs,exp,log
error = lambda i : fabs(exp(-5*euler_method_round(a811,1.0,2**i)[0][-1])
```

```
                            - euler_method_round(a811,1.0,2**i)[1][-1])
y=[log(roundoff(error(i)),2) for i in range(0,-14,-1)]
plt.plot(x,y)
plt.show()
plt.close()
```



次に, この Euler 法を更に精度を上げるために以下の台形法を用いる.

方法 1.2.4 (台形法) 台形法は $t_0 < t_1 < t_2 < \cdots < t_n < \cdots$ という間隔 $\Delta_n = t_{n+1} - t_n$ の離散化に対して, 近似解を

$$y_{n+1} = y_n + \frac{1}{2}\{a(t_n, y_n) + a(t_{n+1}, y_{n+1})\}\Delta_n, \quad y_0 = x_0 \tag{1.2.2}$$

によって与える.

この方法は y_{n+1} という不定値が両辺に含まれているため **implicit** な方法と呼ばれる. 一般にはこの不定値は移行することが出来ないので計算が複雑になることがある. 故にそれを修正したのが, 以下の修正台形法である.

方法 1.2.5 (修正台形法) 修正台形法は $t_0 < t_1 < t_2 < \cdots < t_n < \cdots$ という間隔 $\Delta_n = t_{n+1} - t_n$ の離散化に対して, 近似解を

$$\bar{y}_{n+1} = y_n + a(t_n, y_n)\Delta_n \tag{1.2.3}$$

$$y_{n+1} = y_n + \frac{1}{2}\{a(t_n, y_n) + a(t_{n+1}, \bar{y}_{n+1})\}\Delta_n \tag{1.2.4}$$

によって与える. つまり, $\bar{y}_{n+1}$ を下の式に代入して,

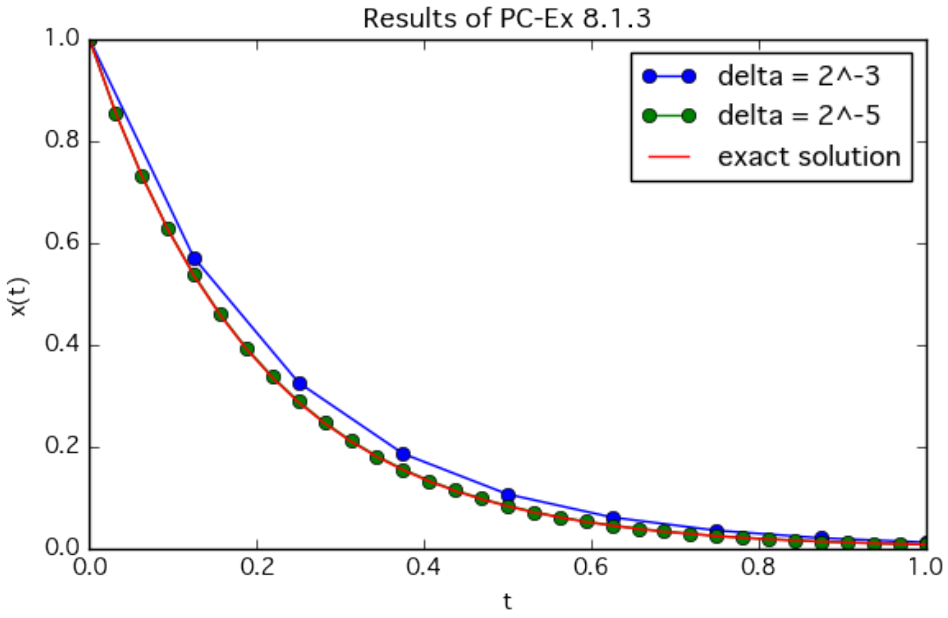
$$y_{n+1} = y_n + \frac{1}{2}\{a(t_n, y_n) + a(t_{n+1}, y_n + a(t_n, y_n)\Delta_n)\}\Delta_n \tag{1.2.5}$$

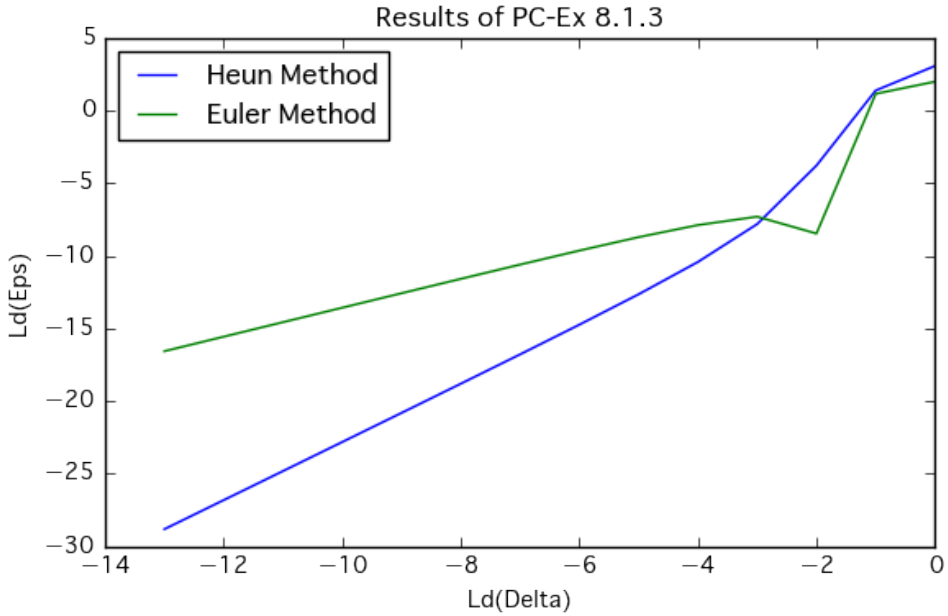
によって与えられる.

修正台形法は,**improved Euler 法**や **Heun 法**とも呼ばれている. また, このような方法は**予測子修正子法**と呼ばれている. というのは, 一度 (1.2.3) で予測子 $\bar{y}_{n+1}$ を計算し, それを (1.2.4) のような式に代入して修正子 y_{n+1} を得ているからである.

例 1.2.6 例 *1.2.2* と同じ微分方程式の初期値問題に対して, 修正台形法の $t = 1$ での大域離散化誤差を計算する. 離散化の間隔は同様に $\Delta = 1, 2^{-1}, 2^{-2}, \cdots, 2^{-13}$ とする.

In[]:
```
def heun_method(a,x0,delta):
    t =[delta*x for x in range(int(1/delta)+1)]
    y =[x0]
    for tn in t[:-1]:
        y_ = y[-1]+a(tn,y[-1])*delta
        y.append(y[-1]+(a(tn,y[-1])+a(tn,y_))/2*delta)
    return [t,y]

plt.title("Results of PC-Ex 8.1.3")
plt.xlabel("t")
plt.ylabel("x(t)")

ans1 = heun_method(a811,1.0,2**(-3))
ans2 = heun_method(a811,1.0,2**(-5))

exact_x = [x*(2**(-5)) for x in range(2**5 + 1)]
exact_y = [math.exp(-5*x) for x in exact_x]

plt.plot(ans1[0],ans1[1],"-o",label="delta = 2^-3")
plt.plot(ans2[0],ans2[1],"-o",label="delta = 2^-5")
plt.plot(exact_x,exact_y ,label = "exact solution")
plt.legend()
plt.show()
plt.close()

plt.title("Results of PC-Ex 8.1.3")
plt.xlabel("Ld(Delta)")
plt.ylabel("Ld(Eps)")
x=[i for i in range(0,-14,-1)]

from math import fabs,exp,log

error = lambda i : fabs(exp(-5*heun_method(a811,1.0,2**i)[0][-1])
                        - heun_method(a811,1.0,2**i)[1][-1])
heun_y=[log(error(i),2) for i in range(0,-14,-1)]

error = lambda i : fabs(exp(-5*euler_method(a811,1.0,2**i)[0][-1])
                        - euler_method(a811,1.0,2**i)[1][-1])
euler_y=[log(error(i),2) for i in range(0,-14,-1)]

plt.plot(x,heun_y,label="Heun Method")
plt.plot(x,euler_y,label="Euler Method")
plt.legend(loc="upper left")

plt.show()
plt.close()
```





ここで今まで上げたような方法を一般化して次の方法を得る.

方法 1.2.7 (一段法) **一段法**は $t_0 < t_1 < t_2 < \cdots < t_n < \cdots$ という間隔 $\Delta_n = t_{n+1} - t_n$ の離散化に対して, 近似解を

$$y_{n+1} = y_n + \Psi(t_n, y_n, \Delta_n)\Delta_n, \quad y_0 = x_0 \tag{1.2.6}$$

によって与える.$\Psi(t, y, \Delta)$ のことを *increment function* という.

例えば Euler 法 (1.2.1) では $\Psi(t, y, \Delta) = a(t, x)$ であり,Heun 法 (1.2.5) では $\Psi(t, y, \Delta) = \frac{1}{2}\{a(t, y) + a(t + \Delta, y + a(t, y)\Delta)\}$ であった. これまで上げた方法は全て一段法であったが, それとは別の多段法を紹介する.

方法 1.2.8 (Adams-Bashford 法) **Adams-Bashford 法**は $t_0 < t_1 < t_2 < \cdots < t_n < \cdots$ という間隔 $\Delta = t_{n+1} - t_n$ が一定な離散化に対して, 近似解を

$$y_{n+1} = y_n + \frac{1}{12}\{23a(t_n, y_n) - 16a(t_{n-1}, y_{n-1}) + 5a(t_{n-2}, y_{n-2})\}\Delta \tag{1.2.7}$$

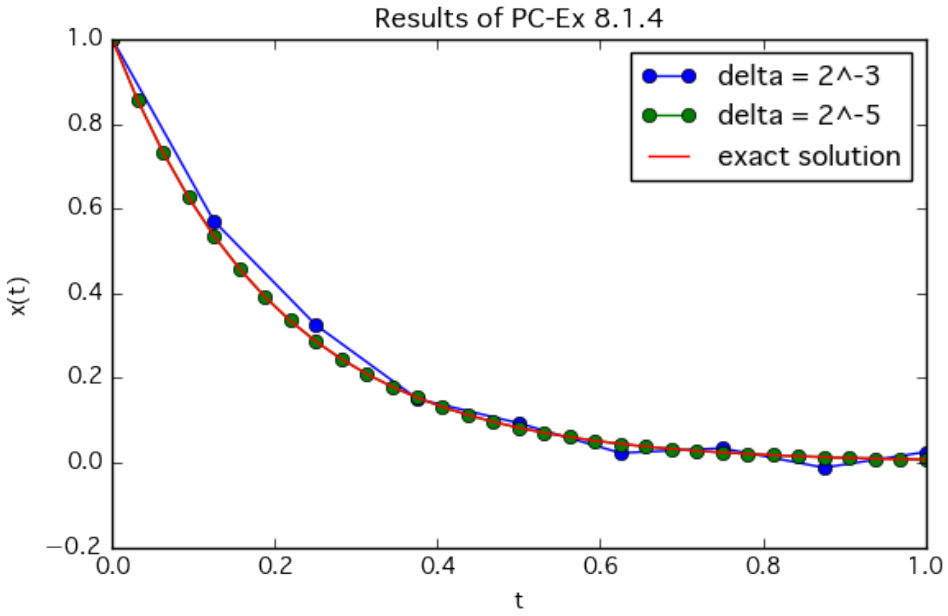
によって与える.

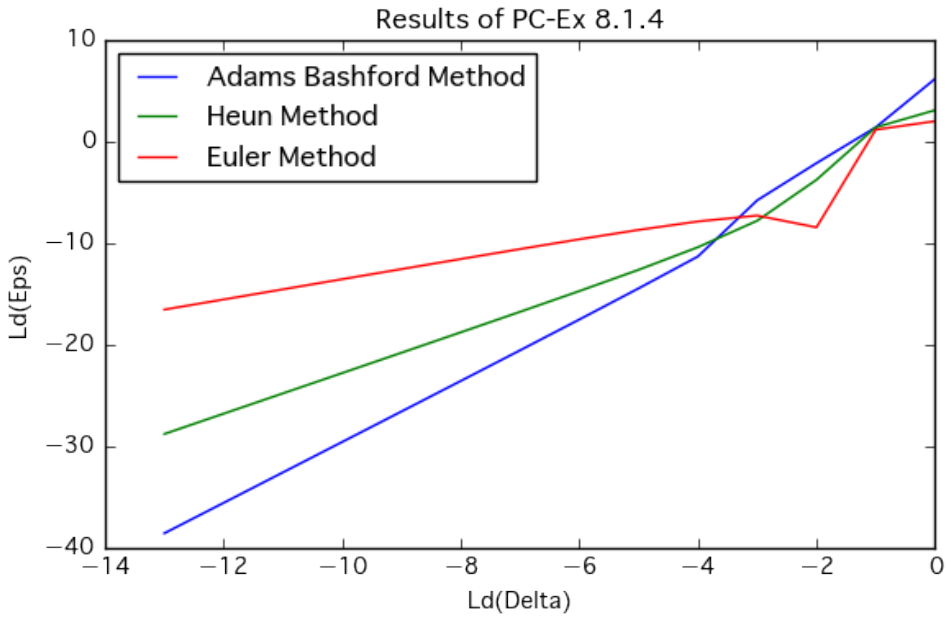
これは,y_{n+1} を求めるために $y_n, y_{n-1}, \cdots, y_{n-k}$ を必要としているために多段法と呼ばれている. 例えば Adams-Bashford 法は 3 段法である.3 段法において, 計算を始めるためには,y_0, y_1, y_2 を求める必要があるが, これらは一段法によって計算される.

例 1.2.9 例 *1.2.2* と同じ微分方程式の初期値問題に対して,*Adams-Bashford* 法の $t = 1$ での大域離散化誤差を計算する. 離散化の間隔は同様に $\Delta = 1, 2^{-1}, 2^{-2}, \cdots, 2^{-13}$ として, 最初の項は修正台形法でもって計算する.

In[]:
```
def adams_bashford_method(a,x0,delta):
    t =[delta*x for x in range(int(1/delta)+1)]
    y =[x0]
    for tn in t[:2]:
        y_ = y[-1]+a(tn,y[-1])*delta
        y.append(y[-1]+(a(tn,y[-1])+a(tn,y_))/2*delta)

    for tn in t[2:-1]:
        y.append(y[-1] + (23*a(tn,y[-1]) - 16*a(tn-delta,y[-2]) +  5*a(tn-2*delta,y[

    return [t,y]

plt.title("Results of PC-Ex 8.1.4")
plt.xlabel("t")
plt.ylabel("x(t)")

ans1 = adams_bashford_method(a811,1.0,2**(-3))
ans2 = adams_bashford_method(a811,1.0,2**(-5))

exact_x = [x*(2**(-5)) for x in range(2**5 + 1)]
exact_y = [math.exp(-5*x) for x in exact_x]

plt.plot(ans1[0],ans1[1],"-o",label="delta = 2^-3")
plt.plot(ans2[0],ans2[1],"-o",label="delta = 2^-5")
plt.plot(exact_x,exact_y ,label = "exact solution")
plt.legend()
plt.show()
plt.close()

plt.title("Results of PC-Ex 8.1.4")
plt.xlabel("Ld(Delta)")
plt.ylabel("Ld(Eps)")
x=[i for i in range(0,-14,-1)]

from math import fabs,exp,log
error = lambda i : fabs(exp(-5*adams_bashford_method(a811,1.0,2**i)[0][-1])
                        - adams_bashford_method(a811,1.0,2**i)[1][-1])
adams_bashford_y=[log(error(i),2) for i in range(0,-14,-1)]

error = lambda i : fabs(exp(-5*heun_method(a811,1.0,2**i)[0][-1])
                        - heun_method(a811,1.0,2**i)[1][-1])
heun_y=[log(error(i),2) for i in range(0,-14,-1)]

error = lambda i : fabs(exp(-5*euler_method(a811,1.0,2**i)[0][-1])
                        - euler_method(a811,1.0,2**i)[1][-1])
euler_y=[log(error(i),2) for i in range(0,-14,-1)]

plt.plot(x,adams_bashford_y,label="Adams Bashford Method")
```

```
plt.plot(x,heun_y,label="Heun Method")
plt.plot(x,euler_y,label="Euler Method")

plt.legend(loc="upper left")

plt.show()
plt.close()
```





また更に高い精度を得るために次のように, 既存の方法を反復するような方法を紹介する.

方法 1.2.10 (Richardson 反復法) 間隔 $\Delta = T/N$ によって $[0,T]$ が等間隔に N 等分されており, その離散化に対して *Euler* 法を適用する場合を考える. $y_N(\Delta)$ を Δ 間隔で離散化したときの近似解の ΔN での

値とする. $x(T)$ を T での真の解の値とする. このとき

$$y_N(\Delta) = x(T) + e(T)\Delta + O(\Delta^2) \tag{1.2.8}$$

が成り立っており, また,$2N$ 等分して離散化したときのことを考えると,

$$y_{2N}(\frac{1}{2}\Delta) = x(T) + \frac{1}{2}e(T)\Delta + O(\Delta^2) \tag{1.2.9}$$

が成り立っている.$e(T)$ を消去することによって,

$$x(T) = 2y_{2N}(\frac{1}{2}\Delta) - y_N(\Delta) + O(\Delta^2) \tag{1.2.10}$$

を得るので,

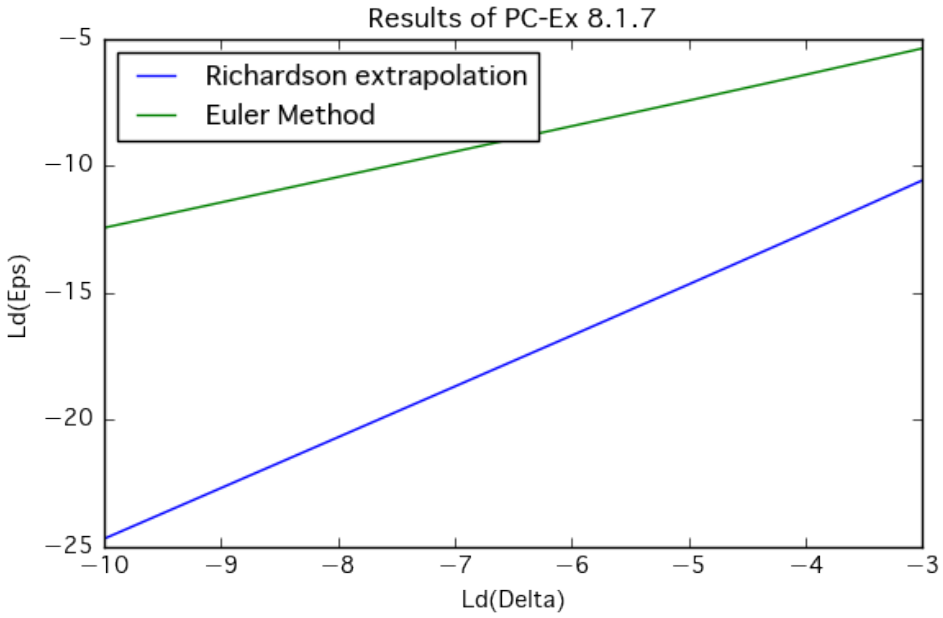
$$Z_N(\Delta) = 2y_{2N}(\frac{1}{2}\Delta) - y_N(\Delta) \tag{1.2.11}$$

とすることによって近似を得る. これを *Richardson* 反復法という.

例 1.2.11 *Richardson* 反復法と *Euler* 法の $t = 1$ での誤差を次の微分方程式の初期値問題に対して比べる.

$$\frac{dx}{dt} = -x,\ x(0) = 1$$

In[]:
```
def a817(t,x):
    return -x

def richardson_extrapolation(a,x0,delta,method):
    yN = method(a,x0,delta)[1][-1]
    y2N = method(a,x0,delta/2)[1][-1]
    return 2*y2N - yN


plt.title("Results of PC-Ex 8.1.7")
plt.xlabel("Ld(Delta)")
plt.ylabel("Ld(Eps)")
x=[i for i in range(-3,-11,-1)]


error = lambda i : fabs(exp(-1) - euler_method(a817,1.0,2**i)[1][-1])
euler_y=[log(error(i),2) for i in range(-3,-11,-1)]

error = lambda i : fabs(exp(-1) - richardson_extrapolation(a817,1.0,2**i,euler_metho
richardson_y =[log(error(i),2) for i in range(-3,-11,-1)]

plt.plot(x,richardson_y,label="Richardson extrapolation")
plt.plot(x,euler_y,label="Euler Method")

plt.legend(loc="upper left")

plt.show()
plt.close()
```



ここで,a に条件を課せば, 次の Taylor の定理が成り立っていることに着目する.

定理 1.2.12 (Taylor の定理) $x(t)$ は $p+1$ 回連続微分可能であるとする. このとき, $t_0 < t_1 < t_2 < \cdots < t_n < \cdots$ という間隔 $\Delta_n = t_{n+1} - t_n$ の離散化に対して,

$$x(t_{n+1}) = x(t_n) + \frac{dx}{dt}(t_n)\Delta_n + \cdots + \frac{1}{p!}\frac{d^p x}{dt^p}(t_n)\Delta_n^p + \frac{1}{(p+1)!}\frac{d^{p+1}x}{dt^{p+1}}(\theta_n)\Delta_n^{p+1} \tag{1.2.12}$$

を満たすような $t_n < \theta_n < t_{n+1}$ が存在する.

そして, 微分方程式

$$\dot{x} = \frac{dx}{dt} = a(t, x(t)) \tag{1.2.13}$$

に着目して, チェインルールを適応すれば,

$$\frac{dx}{dt} = a, \frac{d^2x}{dt^2} = a_t + a_x a, \frac{d^3x}{dt^3} = a_{tt} + 2a_{tx}a + a_{xx}a^2 + a_t a_x + a_x^2 a, \cdots \tag{1.2.14}$$

(t, x での偏微分を省略して a_t, a_x と書いた) が成り立っているので, これらを代入して, 剰余項を無視すれば次のような Taylor 近似が得られる.

方法 1.2.13 (Taylor 近似) p 次 Taylor 近似は *(1.2.13)* をみたす微分方程式と $t_0 < t_1 < t_2 < \cdots < t_n < \cdots$ という間隔 $\Delta_n = t_{n+1} - t_n$ の離散化に対して, 近似解を

$$y_{n+1} = y_n + a(t_n, y_n)\Delta_n + \frac{1}{2!}\frac{da}{dt}(t_n, y_n)\Delta_n^2 + \cdots + \frac{1}{p!}\frac{d^{p-1}a}{dt^{p-1}}(t_n)\Delta_n^p \tag{1.2.15}$$

によって与える.

例 1.2.14 例えば,2 次 *Taylor* 近似は,

$$y_{n+1} = y_n + a(t_n, y_n)\Delta_n + \frac{1}{2!}\{a_t + a_x a\}\Delta_n^2 \tag{1.2.16}$$

3 次 *Taylor* 近似は,

$$y_{n+1} = y_n + a(t_n, y_n)\Delta_n + \frac{1}{2!}\{a_t + a_x a\}\Delta_n^2 + \frac{1}{3!}\{a_{tt} + 2a_{tx}a + a_{xx}a^2 + a_t a_x + a_x^2 a\}\Delta_n^3 \qquad (1.2.17)$$

によって与えられる. 各々の偏微分には (t_n, y_n) を代入する.

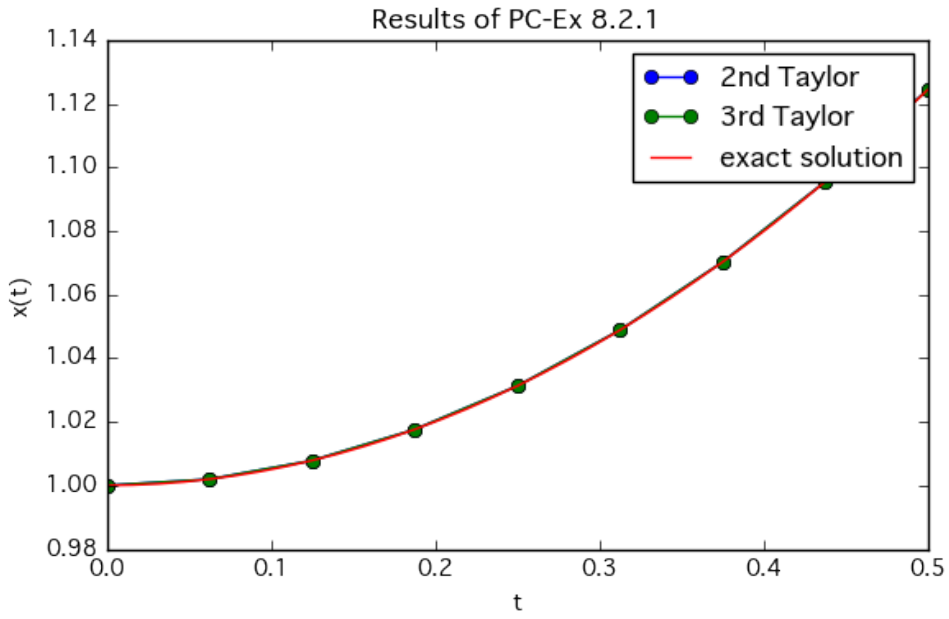
例 1.2.15 2 次および 3 次の *Taylor* 近似を以下の微分方程式の初期値問題に対して適応する. ただし,$\Delta = 2^{-3}, 2^{-2}, \cdots, 2^{-10}$ である.

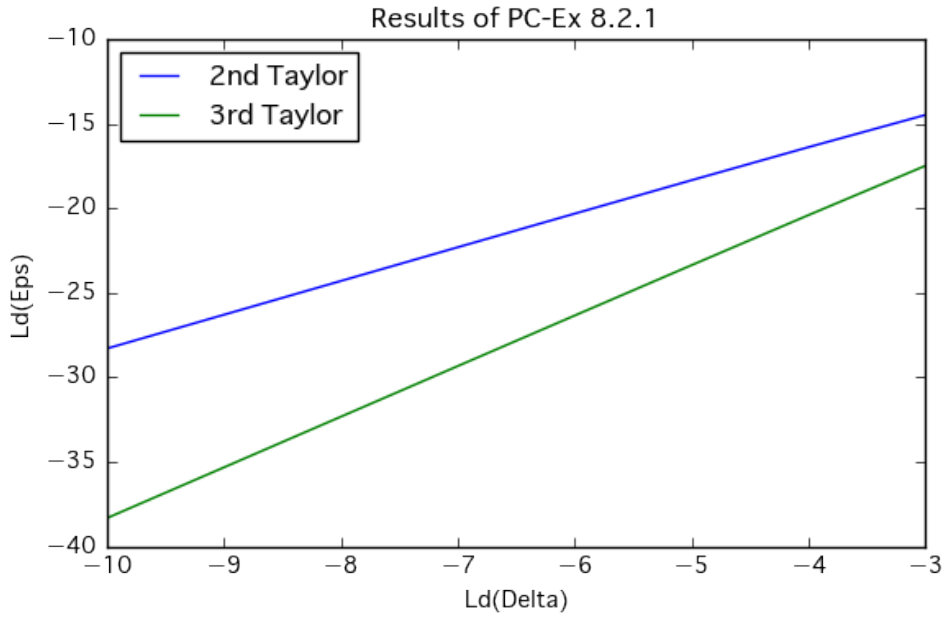
$$\frac{dx}{dt} = tx(2-x),\ x(0) = -1\ \ (0 \le t \le 0.5).$$

In[]:

```
def a821(t,x):
    return t*x*(2-x)

def sol821(t):
    return 2/(1+exp(-(t**2)))

def a_t821(t,x):
    return x*(2-x)

def a_x821(t,x):
    return 2*t*(1-x)

def a_tt821(t,x):
    return 0

def a_tx821(t,x):
    return 2-2*x

def a_xx821(t,x):
    return -2*t

def taylor_2nd(x0,t,a,a_t,a_x):
    y =[x0]
    delta = t[1]-t[0]
    for tn in t[:-1]:
        y.append(y[-1]+a(tn,y[-1])*delta + (a_t(tn,y[-1])+a_x(tn,y[-1])*a(tn,y[-1]))
    return [t,y]

def taylor_3rd(x0,t,a,a_t,a_x,a_tt,a_tx,a_xx):
    y =[x0]
    delta = t[1]-t[0]
    for tn in t[:-1]:
        y.append(y[-1]+
                 a(tn,y[-1])*delta +
                 (a_t(tn,y[-1])+a_x(tn,y[-1])*a(tn,y[-1]))*delta*delta/2 +
                (a_tt(tn,y[-1])+2*a_tx(tn,y[-1])*a(tn,y[-1])+a_xx(tn,y[-1])*a(tn,y[-
                 + a_x(tn,y[-1])*a_x(tn,y[-1])*a(tn,y[-1]))*delta*delta*delta/6)

    return [t,y]

plt.title("Results of PC-Ex 8.2.1")
plt.xlabel("t")
plt.ylabel("x(t)")

delta = 2**(-4)
t =[delta*x for x in range(int(0.5/delta)+1)]
ans1 = taylor_2nd(1.0,t,a821,a_t821,a_x821)
ans2 = taylor_3rd(1.0,t,a821,a_t821,a_x821,a_tt821,a_tx821,a_xx821)

exact_x = [x*(2**(-6)) for x in range(2**5 + 1)]
exact_y = [sol821(x) for x in exact_x]

plt.plot(ans1[0],ans1[1],"-o",label="2nd Taylor")
plt.plot(ans2[0],ans2[1],"-o",label="3rd Taylor")
plt.plot(exact_x,exact_y ,label = "exact solution")
plt.legend()
plt.show()
plt.close()
```

```
plt.title("Results of PC-Ex 8.2.1")
plt.xlabel("Ld(Delta)")
plt.ylabel("Ld(Eps)")
x=[i for i in range(-3,-11,-1)]

error = lambda i : fabs(sol821(0.5)  - taylor_2nd(1.0,list(np.linspace(0,0.5,2**(-i)
taylor_2nd_error = [log(error(i),2) for i in x]

error = lambda i : fabs(sol821(0.5)  - taylor_3rd(1.0,list(np.linspace(0,0.5,2**(-i)
taylor_3rd_error = [log(error(i),2) for i in x]

plt.plot(x,taylor_2nd_error,label="2nd Taylor")
plt.plot(x,taylor_3rd_error,label="3rd Taylor")

plt.legend(loc="upper left")

plt.show()
plt.close()
```





しかし, これらの Taylor 近似は, 与えられた a に対して偏微分を計算しなければならないという点において実用的ではない. その場合には,

$$a_t(t_n, y_n) \approx \frac{a(t_{n+1}, y_n) - a(t_n, y_n)}{\Delta_n}, a_x(t_n, y_n) \approx \frac{a(t_n, y_{n+1}) - a(t_n, y_n)}{y_{n+1} - y_n} \tag{1.2.18}$$

という近似を使って偏微分を計算すれば良い. また,a_x の計算には y_{n+1} が現れるが, それは Euler 法等により予測すれば良い.

再び一段法について着目する. 一段法はヒューリスティックに

$$y_{n+1} = y_n + \Psi(t_n, y_n, \Delta_n)\Delta_n \tag{1.2.19}$$

という式の $\Psi(t, y, \Delta)$ を計算することが重要であったが, ここではその方法の典型的な例である Runge-Kutta 法について触れる.

方法 1.2.16 (2 次 Runge-Kutta 法) 2 次 Runge-Kutta 法は

$$y_{n+1} = y_n + \Psi(t_n, y_n, \Delta_n)\Delta_n, \ \ y_0 = x_0 \tag{1.2.20}$$

という一段法の *increment function* に対して,

$$\Psi(t, y, \Delta) = \alpha\, a(t, x) + \beta\, a(t + \gamma\Delta, x + \gamma a(t, x)\Delta) \tag{1.2.21}$$

を代入することによって得られる.

ここで, 右辺を展開することによって,

$$\Psi(t, y, \Delta) = (\alpha + \beta)a(t, x) + \gamma\beta(a_t + a_x a)\Delta + \frac{1}{2}\gamma^2\beta(a_{tt} + 2a_{tx}a + a_{xx}a^2)\Delta^2 + \cdots$$

が得られる. ここで 3 次 Taylor 近似と各項を比較すると,

$$y_{n+1} = y_n + a(t_n, y_n)\Delta_n + \frac{1}{2!}\{a_t + a_x a\}\Delta_n^2 + \frac{1}{3!}\{a_{tt} + 2a_{tx}a + a_{xx}a^2 + a_t a_x + a_x^2 a\}\Delta_n^3$$

$$\alpha + \beta = 1, \ \ \gamma\beta = \frac{1}{2}$$

が得られる. 一方で $\alpha = \beta = 1/2, \gamma = 1$ を代入するとこれは Heun 法 (1.2.5) となる. ここで更に 4 次の Runge-Kutta 法について紹介する.

方法 1.2.17 (4 次 Runge-Kutta 法) 4 次 Runge-Kutta 法は $t_0 < t_1 < t_2 < \cdots < t_n < \cdots$ という間隔 $\Delta_n = t_{n+1} - t_n$ の離散化に対して, 近似解を

$$y_{n+1} = y_n + \frac{1}{6}\{k_n^{(1)} + 2k_n^{(2)} + 2k_n^{(3)} + k_n^{(4)}\}\Delta_n \tag{1.2.22}$$

によって与える. ただし,

$$k_n^{(1)} = a(t_n, y_n), \tag{1.2.23}$$

$$k_n^{(2)} = a\left(t_n + \frac{1}{2}\Delta_n, y_n + \frac{1}{2}k_n^{(1)}\Delta_n\right), \tag{1.2.24}$$

$$k_n^{(3)} = a\left(t_n + \frac{1}{2}\Delta_n, y_n + \frac{1}{2}k_n^{(2)}\Delta_n\right), \tag{1.2.25}$$

$$k_n^{(4)} = a\left(t_{n+1}, y_n + k_n^{(3)}\Delta_n\right) \tag{1.2.26}$$

4 次の Runge-Kutta 法は,

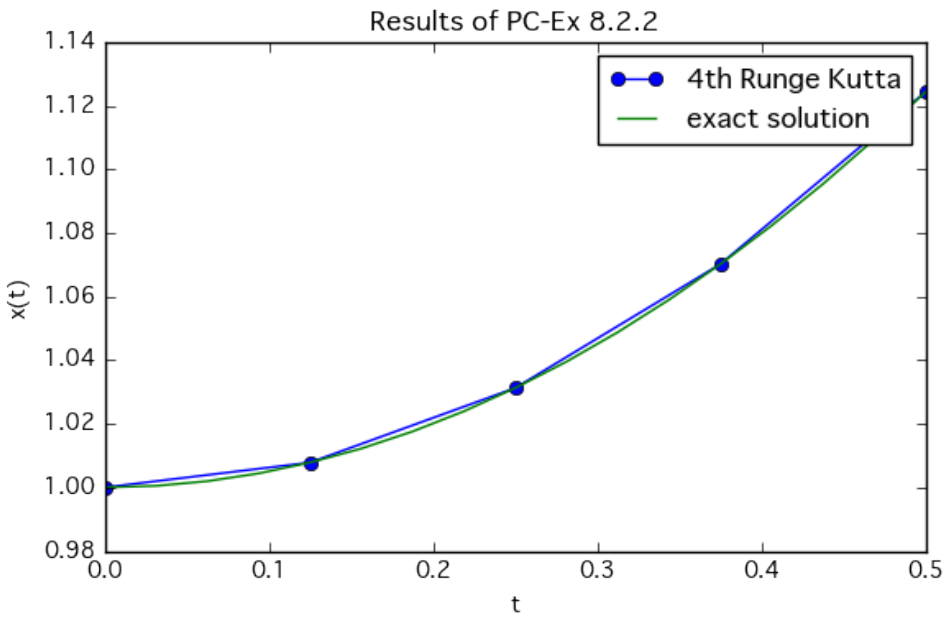
$$y_{n+1} - y_n = \int_{t_n}^{t_{n+1}} a(t)dt$$

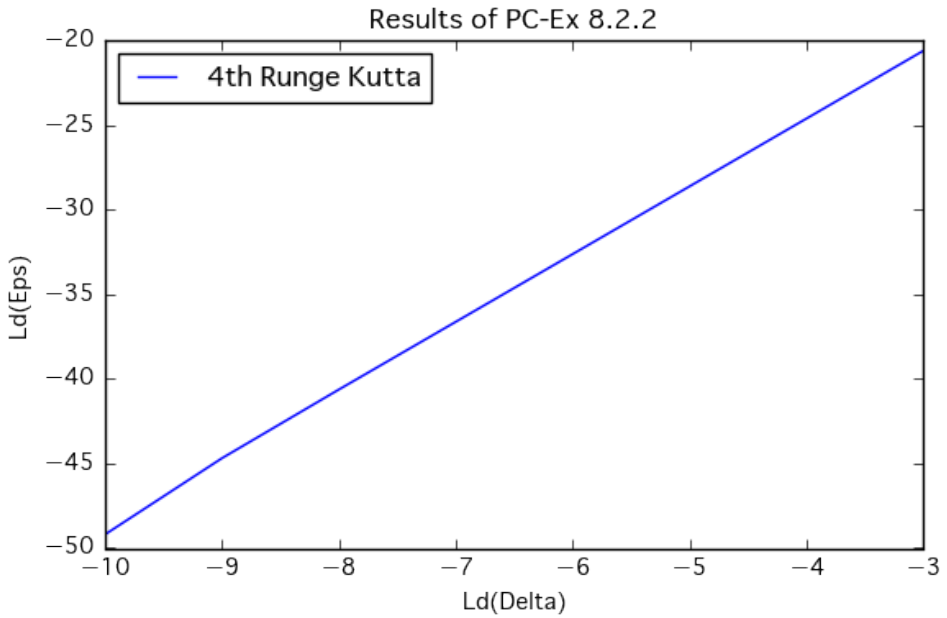
に対して Simpson の公式を当てはめて積分を計算しただけである.

例 1.2.18 4 次の *Runge-Kutta* 法を以下の微分方程式の初期値問題に対して適応する. ただし,$\Delta = 2^{-3}, 2^{-2}, \cdots, 2^{-10}$ である.

$$\frac{dx}{dt} = tx(2-x),\ x(0) = -1\ \ (0 \le t \le 0.5).$$

In[]:
```
def runge_kutta_4th(x0,t0,tn,delta,a):
    t = list(np.arange(t0,tn+delta,delta))
    y = [x0]
    for tn in t[:-1]:
        k1 = a(tn,y[-1])
        k2 = a(tn+delta/2, y[-1]+k1*delta/2)
        k3 = a(tn+delta/2, y[-1]+k2*delta/2)
        k4 = a(tn+delta, y[-1]+k3*delta)
        y.append(y[-1]+(k1+2*k2+2*k3+k4)*delta/6)

    return[t,y]

plt.title("Results of PC-Ex 8.2.2")
plt.xlabel("t")
plt.ylabel("x(t)")


ans1 = runge_kutta_4th(1.0,0,0.5,2**(-3),a821)
exact_x =  list(np.arange(0,0.5+2**(-5),2**(-5)))
exact_y = [sol821(x) for x in exact_x]

plt.plot(ans1[0],ans1[1],"-o",label="4th Runge Kutta")
plt.plot(exact_x,exact_y ,label = "exact solution")
plt.legend()
plt.show()
plt.close()

plt.title("Results of PC-Ex 8.2.2")
plt.xlabel("Ld(Delta)")
plt.ylabel("Ld(Eps)")
x=[i for i in range(-3,-11,-1)]

error = lambda i : fabs(sol821(0.5)  - runge_kutta_4th(1.0,0,0.5,2**(i),a821)[1][-1]
runge_kutta_4th_error = [log(error(i),2) for i in x]

plt.plot(x,runge_kutta_4th_error,label="4th Runge Kutta")

plt.legend(loc="upper left")

plt.show()
plt.close()
```





次に多段法についていくつかの例を紹介する.

方法 1.2.19 (一般の多段法) **多段法**は $t_0 < t_1 < t_2 < \cdots < t_n < \cdots$ という間隔 $\Delta = t_{n+1} - t_n$ が一定な離散化に対して, 近似解を

$$y_{n+1} = \sum_{j=1}^{k} \alpha_j y_{n+1-j} + \sum_{j=0}^{k} \beta_j a(t_{n+1-j}, y_{n+1-j})\Delta \tag{1.2.27}$$

によって与える. ただし α_j, β_j は定数である.

定義 1.2.20 式 *(1.2.27)* において,$\beta_0 = 0$ のとき, これを *explicit* な方法と呼び, $\beta_0 \neq 0$ のとき, これを *implicit* な方法と呼ぶ.

例 1.2.21 多段法の例を 3 つ上げる. それぞれ近似解は次の式で与えられる.

中点法

$$y_{n+1} = y_{n-1} + 2a(t_n, y_n)\Delta \tag{1.2.28}$$

Milne 法

$$y_{n+1} = y_{n-3} + \frac{4}{3}\{2a(t_n, y_n) - a(t_{n-1}, y_{n-1}) + 2a(t_{n-2}, y_{n-2})\}\Delta \tag{1.2.29}$$

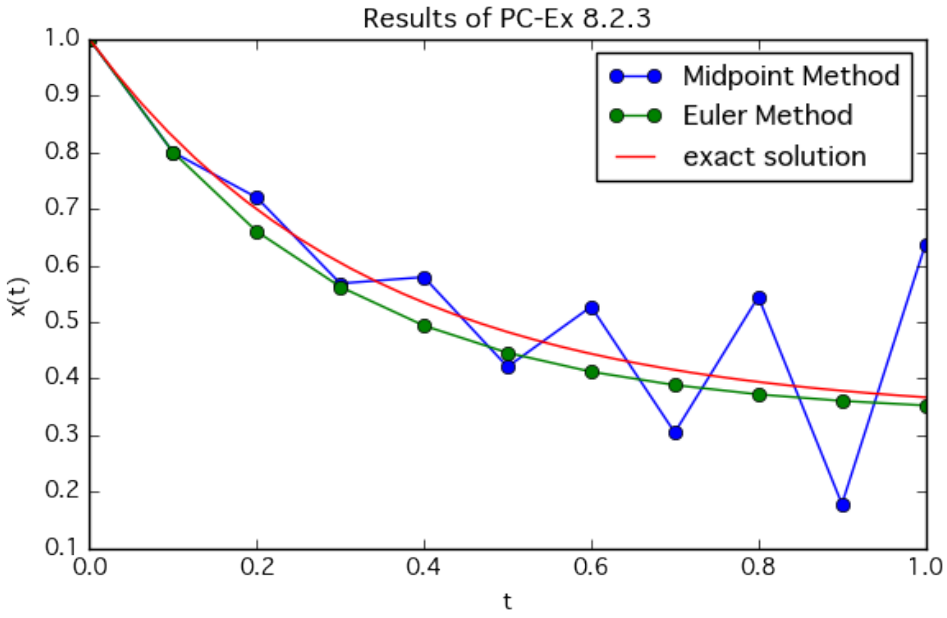
Adams-Moulton 法

$$y_{n+1} = y_n + \frac{1}{12}\{5a(t_{n+1}, y_{n+1}) + 8a(t_n, y_n) - a(t_{n-1}, y_{n-1})\}\Delta \tag{1.2.30}$$

例 1.2.22 *Euler* 法と中点法を次の微分方程式の初期値問題に対して適応する. ただし,$\Delta = 0.1$ とする.

$$\frac{dx}{dt} = -3x + 1,\ x(0) = 1(0 \leq t \leq 1).$$

In[]:
```
def a823(t,x):
    return (-3)*x +1

def sol823(t):
    return 2*exp((-3)*t)/3 + 1/3

def midpoint_method(x0,t0,t1,delta,a):
    t = list(np.arange(t0,t1+delta,delta))
    y = [x0,x0+a(t[0],x0)*delta]

    for tn in t[1:-1]:
            y.append(y[-2]+2*a(tn,y[-1])*delta)

    return[t,y]

plt.title("Results of PC-Ex 8.2.3")
plt.xlabel("t")
plt.ylabel("x(t)")

ans1 = midpoint_method(1,0,1,0.1,a823)
ans2 = euler_method(a823,1.0,0.1)

exact_x = [x*(2**(-5)) for x in range(2**5 + 1)]
exact_y = [sol823(x) for x in exact_x]

plt.plot(ans1[0],ans1[1],"-o",label="Midpoint Method")
plt.plot(ans2[0],ans2[1],"-o",label="Euler Method")
plt.plot(exact_x,exact_y ,label = "exact solution")
plt.legend()
plt.show()
plt.close()
```



1.3 近似方法の性質

今までに挙げた常微分方程式の近似方法について, 幾つかの性質を上げて, それらの性質のつながりを見る.

定義 1.3.1 微分方程式の初期値問題と $t_0 < t_1 < t_2 < \cdots < t_n < \cdots$ という離散化に対して, ある近似方法によって $y_0, y_1, \cdots, y_n, \cdots$ という近似解が得られたとする. ここで,$x(t; t_n, y_n)$ を $x(t_n) = y_n$ という初期値問題の解とおく. このとき,

$$l_{n+1} := x(t_{n+1}; t_n, y_n) - y_{n+1} \tag{1.3.1}$$

$$e_{n+1} := x(t_{n+1}; t_0, x_0) - y_{n+1} \tag{1.3.2}$$

として,l_{n+1}, e_{n+1} をそれぞれ**局所・大域 (離散化) 誤差**と呼ぶ.

ここで,Euler 法 (1.2.1)

$$y_{n+1} = y_n + a(t_n, y_n)\Delta_n, \quad y_0 = x_0$$

の離散化誤差を幾つかの不等式評価によって評価することを考える.
まず, 局所離散化誤差 l_{n+1} の評価をする. これは,$x(t)$ が C^2 級であることを仮定すると以下のテイラーの定理により,

$$x(t_{n+1}) = x(t_n) + \dot{x}(t_n)\Delta_n + \frac{1}{2!}\ddot{x}(\theta_n)\Delta_n^2$$

を満たすような $\theta_n \in [t_n, t_{n+1}]$ が存在する. ここで,

$$\dot{x}(t_n) = a(t_n, x(t_n)), \ x(t_n) = y_n$$

ということに着目すると,

$$x(t_{n+1}; t_n, y_n) = x(t_n) + a(t_n, x(t_n))\Delta_n + \frac{1}{2!}\ddot{x}(\theta_n)\Delta_n^2$$

により,

$$l_{n+1} = \frac{1}{2!}\ddot{x}(\theta_n)\Delta_n^2 \tag{1.3.3}$$

を得る. ここで,$|\ddot{x}(t)| < M$ なる $M > 0$ が存在すれば,

$$|l_{n+1}| \leq \frac{1}{2!}M\Delta_n^2 \tag{1.3.4}$$

であるという評価が得られる. ここで,

$$\ddot{x} = a_t + a_x a$$

であることを考えると,

$$M = \max|a_t| + \max|a_x a|$$

であることが分かる. 今, 微分方程式を有界閉領域上で考えているとするとこのような M は (かなり無駄な不等式評価となるが, 有限で) 必ず存在するので,Euler 法の局所離散化誤差は Δ_n^2 のオーダーであることが分かる.

次に Euler 法の大域離散化誤差 l_{n+1} の評価について考える.
まず,$a = a(t,x)$ は一様 Lipschitz 条件を満たしており,

$$|a(t,x) - a(t,y)| \leq K|x-y|$$

をみたすような K が一様に存在しているとする.
また簡単のために間隔 $\Delta_n = t_{n+1} - t_n$ は一定で Δ であるとする. ここで再び

$$x(t_{n+1}) = x(t_n) + \dot{x}(t_n)\Delta_n + \frac{1}{2!}\ddot{x}(\theta_n)\Delta_n^2$$

というテイラーの定理に対して,$y_{n+1} = y_n + a(t_n, y_n)\Delta$ を両辺から引くことによって,

$$e_{n+1} = e_n + (a(t_n, x(t_n)) - a(t_n, y_n))\Delta + \frac{1}{2!}\ddot{x}(\theta_n)\Delta^2$$

が得られ,Lipschitz 条件を考えると, 両辺の絶対値は,

$$|e_{n+1}| \leq |e_n| + K|e_n|\Delta + \frac{1}{2!}M\Delta^2$$

という形で不等式評価することができる. これを帰納的に用いることによって, 最終的に $e_0 = x_0 - y_0 = 0$ になるので,

$$|e_{n+1}| \leq \frac{1}{2}\left(\frac{(1+K\Delta)^n - 1}{(1+K\Delta) - 1}\right)M\Delta^2 \leq \frac{1}{2}(e^{nK\Delta} - 1)\frac{M}{K}\Delta$$

であるので,Euler 法の大域離散化誤差は Δ のオーダーであることが分かる.

次に一般の一段法について整合性, 安定性, 収束といった性質を紹介する.
一段法とは,

$$\dot{x} = \frac{dx}{dt} = a(t,x),\ \ x(t_0) = x_0 \tag{1.3.5}$$

という微分方程式の初期値問題に対して

$$y_{n+1} = y_n + \Psi(t_n, y_n, \Delta_n)\Delta_n,\ \ y_0 = x_0 \tag{1.3.6}$$

という式によって近似解を求める方法であった.

例 1.3.2 *Heun* 法の局所離散化誤差は Δ^3 のオーダーである.
$y(t_n) = y_n$ という初期値問題の解を $y(t)$ とおき,*Heun* 法により y_{n+1} が求まったとする.

$A(t) = a(t, y(t))$ とおくと

$$
\begin{aligned}
y(t_{n+1}) - y_n &= \int_{t_n}^{t_{n+1}} A(s)ds && (1.3.7)\\
&= \int_{t_n}^{t_{n+1}} (A(t_n) + A'(t_n)(s-t_n) + \frac{1}{2}A''(\theta_n(s))(s-t_n)^2 ds && (1.3.8)\\
&= a(t_n, y_n)\Delta_n + \frac{1}{2}A'(t_n)\Delta_n^2 + \int_{t_n}^{t_{n+1}} \frac{1}{2}A''(\theta_n(s))(s-t_n)^2 ds && (1.3.9)\\
&= \frac{1}{2}a(t_n, y_n)\Delta_n + \frac{1}{2}(a(t_n, y_n) + A'(t_n)\Delta_n)\Delta_n + \int_{t_n}^{t_{n+1}} \frac{1}{2}A''(\theta_n(s))(s-t_n)^2 ds && (1.3.10)
\end{aligned}
$$

により, ここでテイラーの定理より,

$$a(t_n, y_n) = a(t_{n+1}, y_{n+1}) - A'(t_n)\Delta_n - \frac{1}{2!}A''(\theta_n)\Delta_n^2$$

なる $\theta_n \in [t_n, t_{n+1}]$ を取ることができて, 局所離散化誤差は $y(t_{n+1}) - y_n - \frac{1}{2}(a(t_n, y_n) + a(t_{n+1}, y_{n+1}))\Delta_n$

$$= -\frac{1}{2^2}A''(\theta_n)\Delta_n^3 + \frac{1}{2}\int_{t_n}^{t_{n+1}} A''(\theta_n(s))(s-t_n)^2 ds$$

となって 3 次のオーダーであることがわかった.

定義 1.3.3 一段法 *(1.3.6)* が微分方程式 *(1.3.5)* に対して**整合的**であるとは,

$$\Psi(t, y, 0) = a(t, x)$$

をみたすということである

定義 1.3.4 $\Delta = \max_n \Delta_n$ とおく. 一段法 *(1.3.6)* が**収束**するとは, 任意の有界区間 $[t_0, T]$ において,Δ を 0 に近づけると, 大域離散化誤差 e_{n+1} が 0 に収束するということであり, つまり

$$\lim_{\Delta \downarrow 0} |e_{n+1}| = 0$$

をみたすということである

定義 1.3.5 一段法 *(1.3.6)* が**数値的に安定**であるとは, 任意の区間 $[t_0, T]$ と任意の *Lipschitz* 条件を満たす $a(t, x)$ に対して, ある Δ_0 と M という正の定数が存在して,$y_n, \tilde{y}_n$ という $\Delta = \max_n \Delta_n < \Delta_0$ を満たす離散化の初期値 $y_0, \tilde{y}_0$ に対応する数値解に対して,

$$|y_n - \tilde{y}_n| \leq M|y_0 - \tilde{y}_0|,\ n = 0, 1, \cdots, n_T$$

をみたすということである

定義 1.3.6 一段法 *(1.3.6)* が**漸近的に数値的安定**であるとは, 任意の $a(t, x)$ に対して, ある Δ_a と M という正の定数が存在して,$y_n, \tilde{y}_n$ という任意の $\Delta = \max_n \Delta_n < \Delta_a$ を満たす離散化の数値解に対して,

$$\lim_{n \to \infty} |y_n - \tilde{y}_n| \leq M|y_0 - \tilde{y}_0|,\ n = 0, 1, \cdots, n_T$$

をみたすということである

ここで数値的安定性に対して, 更に新しい考え方を導入する.

定義 1.3.7 一段法 *(1.3.6)* の**絶対安定領域**とは次のような複素平面上の領域 R である.
$\lambda\Delta \in R \Leftrightarrow$

$$\frac{dx}{dt} = \lambda x$$

という複素微分方程式を考え, 間隔 Δ の離散化に対する数値解 y_n が任意の初期値 x_0 に対して

$$\lim_{n\to\infty} y_n = 0$$

を満たす

例 1.3.8 *Euler* 法の絶対安定領域を考える.

$$\frac{dx}{dt} = \lambda x$$

に対して,*Euler* 法を実行すると,

$$y_n = (1 + \lambda\Delta)y_0$$

が得られる. よって任意の y_0 に対して y_n が収束するためには,

$$|1 + \lambda\Delta| < 1$$

が得られ, これがオイラー法の絶対安定領域である.

例 1.3.9 同様に台形法の絶対安定領域を考える.

$$y_{n+1} = y_n + \frac{1}{2}\lambda\Delta y_n + \frac{1}{2}\lambda\Delta y_{n+1}$$

であるので, これを移行して

$$y_{n+1} = \frac{1 + \frac{1}{2}\lambda\Delta}{1 - \frac{1}{2}\lambda\Delta} y_n$$

であるので, 絶対安定領域は,

$$|1 + \frac{1}{2}\lambda\Delta| < |1 - \frac{1}{2}\lambda\Delta|$$

となり, これは左半平面である.

定義 1.3.10 一段法 *(1.3.6)* が **A 安定**であるとは,
この一段法の絶対安定領域 R が左半平面を含むことつまり,

$$\{z \in \mathbb{C} | \Re z < 0\} \subset R$$

であるということである.

これから, これらの一見無関係に見える近似法の性質の関係を示す.

定理 1.3.11 一段法の *increment function* Ψ が大域 *Lipschitz* 条件を満たし, 大域的に有界であるとする. つまり,

$$|\Psi(t', y', \Delta') - \Psi(t, y, \Delta)| \leq K(|t' - t| + |x' - x| + |\Delta' - \Delta|)$$

を満たすような K が存在し,

$$|\Psi(t, y, 0)| \leq L$$

を満たすような L が存在するとする. このとき, 一段法の収束性と整合性は同値である.

>証明 まず Ψ に対する仮定から大域離散化誤差 e_{n+1} を評価を導く. 微分方程式

$$\frac{dz}{dt} = \Psi(t, z, 0),\ z(t_0) = x_0$$

は Lipschitz 条件から唯一つ解を持つ. この解は連続微分可能であるので, 平均値の定理から,

$$z(t_{n+1}) - z(t_n) = \Psi(t_n + \theta_n\Delta_n\, z(t_n + \theta_n\Delta_n, 0), 0)\Delta_n$$

を満たすような $0 < \theta_n < 1$ が存在する.y_n を初期値 $y_0 = x_0$ に対する一段法の近似解であるとする. つまり, 以下を満たすとする.

$$y_{n+1} = y_n + \Psi(t_n, y_n, \Delta_n)\Delta_n,\ \ y_0 = x_0$$

$\bar{e}_n = y_n - z(t_n)$ として定めると, 辺々を引くことによって,

$$\begin{aligned}
\bar{e}_{n+1} &= \bar{e}_n + (\Psi(t_n, y_n, \Delta_n) - \Psi(t + \theta_n\Delta_n\, z(t_n + \theta_n\Delta_n, 0), 0))\Delta_n && (1.3.11)\\
&= \bar{e}_n + (\Psi(t_n, y_n, \Delta_n) - \Psi(t_n, z(t_n), 0))\Delta_n + (\Psi(t_n, z(t_n), 0) - \Psi(t + \theta_n\Delta_n\, z(t_n + \theta_n\Delta_n, 0), 0))\Delta_n && (1.3.12)
\end{aligned}$$

が得られる. ここで Lipschitz 条件を使うと,

$$|\bar{e}_{n+1}| \le |\bar{e}_n| + K(|\bar{e}_n| + \Delta_n)\Delta_n + K(\theta_n\Delta_n + |z(t_n) - z(t_n + \theta_n\Delta_n, 0)|)\Delta_n$$

が得られる.$|z(t_n) - z(t_n + \theta_n\Delta_n, 0)|$ を最初の平均値の定理によって評価すると,

$$|z(t_n) - z(t_n + \theta_n\Delta_n, 0)| \le |\Psi(t_n + \bar{\theta}_n\theta_n\Delta_n\, z(t_n + \bar{\theta}_n\theta_n\Delta_n, 0), 0)|\theta_n\Delta_n \le L\theta_n\Delta_n$$

とできる. ただし,$0 < \bar{\theta}_n < 1$ これを代入することによって,

$$|\bar{e}_{n+1}| \le (1 + K\Delta)|\bar{e}_n| + K(L + 2)\Delta^2,\ (\Delta = \max_n \Delta_n)$$

を得る. 故にこの不等式を帰納的に使うことによって,

$$|\bar{e}_n| \le (L + 2)(e^{K(T - t_0)} - 1)\Delta$$

が成り立つ. これにより,y_n は $z(t)$ に収束することがわかった.

ここで, 整合性を仮定すると,$z(t)$ は $dx/dt = a(t, x)$ の唯一解であるので,$\bar{e}_n$ がその大域離散化誤差となり収束性が示される.

また, 収束性を仮定すると $[t_0, T]$ 上で常に $z(t) = x(t)$ である. 一方で整合的でないと仮定すると $a(t_0, x_0) \neq \Psi(t_0, x_0, 0)$ となる点がある. これは,

$$\frac{dx}{dt}(t_0) = a(t_0, x_0) \neq \Psi((t_0, x_0), 0) = \frac{dz}{dt}(t_0)$$

により,$z(t) = x(t)$ に矛盾する. □

定理 1.3.12 一段法の *increment function* Ψ が大域 *Lipschitz* 条件を満たすとする.
このとき, 局所離散化誤差 l_{n+1} が Δ^{p+1} のオーダーであるならば, 大域離散化誤差 e_{n+1} は Δ^p のオーダーである.

>証明 この初期値問題の解を $x(t)$ とおき, この一段法による近似解を y_n とおく. 大域離散化誤差は定義から

$$\begin{aligned}
e_{n+1} &= y_{n+1} - x(t_{n+1}) && (1.3.13)\\
&= e_n + \Psi(t_n, y_n, \Delta_n)\Delta_n + x(t_n) - x(t_{n+1}) && (1.3.14)\\
&= e_n + (\Psi(t_n, y_n, \Delta_n) - \Psi(t_n, x(t_n), \Delta_n))\Delta_n + (\Psi(t_n, x(t_n), \Delta_n)\Delta_n + x(t_n) - x(t_{n+1})) && (1.3.15)
\end{aligned}$$

今, 仮定から Ψ は大域 Lipschitz 条件を満たし, 局所離散化誤差,$x(t_{n+1};t_n,y_n)-y_{n+1}$ は Δ^{p+1} のオーダーであるので,

$$|e_{n+1}| \leq |e_n| + K|e_n|\Delta_n + D\Delta^{p+1}$$

が得られ, これを帰納的に繰り返すことによって,$\max_n \Delta_n = \Delta$ とおいて

$$|e_n| \leq \frac{D}{K}(e^{K(T-t_0)}-1)\Delta^p$$

が成り立つ. □

定理 1.3.13 一段法の *increment function* Ψ が大域 *Lipschitz* 条件を満たすならば一段法は数値的安定である.

>証明 Ψ が大域 Lipschitz 条件を満たすと仮定する. $y_n, \tilde{y}_n$ という任意の解に対して,

$$\begin{aligned} |y_{n+1}-\tilde{y}_{n+1}| &\leq |y_n - \tilde{y}_n + \Delta(\Psi(t_n,y_n,\Delta)-\Psi(t_n,\tilde{y}_n,\Delta))| && (1.3.16)\\ &\leq |y_n-\tilde{y}_n| + \Delta|\Psi(t_n,y_n,\Delta)-\Psi(t_n,\tilde{y}_n,\Delta)| && (1.3.17)\\ &\leq (1+K\Delta)|y_n-\tilde{y}_n| && (1.3.18) \end{aligned}$$

をみたすので,$M=(1+K\Delta)^{n_T}$ とおけば数値的安定性が示せる.

□