Chapter.8 常微分方程式の離散時間近似

伊藤克哉

東京大学

2016/09/26

講義ノート　スライド　コードは
https://github.com/KatsuyaITO/NSofSDE

微分方程式の初期値問題

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

$\rightarrow$ 常に解けるとは限らない.
$\rightarrow$ 数値解析を使う.

方法 1.1 (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

$$y_{n+1} = y_n + a(t_n, y_n)\Delta_n, \quad y_0 = x_0 \tag{1.1}$$

によって与える.

方法 1.2 (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

$$y_{n+1} = y_n + \frac{1}{2}\{a(t_n, y_n) + a(t_{n+1}, y_{n+1})\}\Delta_n, \quad y_0 = x_0 \tag{1.2}$$

によって与える.

この方法は両辺に y_{n+1} が含まれている.(implicit)

方法 1.3 (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

$$\bar{y}_{n+1} = y_n + a(t_n, y_n)\Delta_n \quad (1.3)$$

$$y_{n+1} = y_n + \frac{1}{2}\{a(t_n, y_n) + a(t_{n+1}, \bar{y}_{n+1})\}\Delta_n \quad (1.4)$$

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

$$y_{n+1} = y_n + \frac{1}{2}\{a(t_n, y_n) + a(t_{n+1}, y_n + a(t_n, y_n)\Delta_n)\}\Delta_n \quad (1.5)$$

によって与えられる.

improved Euler 法や **Heun 法**とも.
予測子修正子法

方法 1.4 (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

$$y_{n+1} = y_n + \Psi(t_n, y_n, \Delta_n)\Delta_n, \quad y_0 = x_0 \tag{1.6}$$

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

方法 1.5 (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

$$y_{n+1} = y_n + \frac{1}{12}\{23a(t_n, y_n) - 16a(t_{n-1}, y_{n-1}) + 5a(t_{n-2}, y_{n-2})\}\Delta \quad (1.7)$$

によって与える.

多段法

方法 1.6 (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

$$y_N(\Delta) = x(T) + e(T)\Delta + O(\Delta^2) \tag{1.8}$$

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，

$$y_{2N}(\frac{1}{2}\Delta) = x(T) + \frac{1}{2}e(T)\Delta + O(\Delta^2) \tag{1.9}$$

が成り立っている.$e(T)$ を消去することによって，

$$x(T) = 2y_{2N}(\frac{1}{2}\Delta) - y_N(\Delta) + O(\Delta^2) \tag{1.10}$$

を得るので，

$$Z_N(\Delta) = 2y_{2N}(\frac{1}{2}\Delta) - y_N(\Delta) \tag{1.11}$$

定理 1.7 (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,

$$x(t_{n+1}) = x(t_n) + \frac{dx}{dt}(t_n)\Delta_n + \cdots + \frac{1}{p!}\frac{d^p x}{dt^p}(t_n)\Delta_n^p + \frac{1}{(p+1)!}\frac{d^{p+1}x}{dt^{p+1}}(\theta_n)\Delta_n^{p+1} \tag{1.12}$$

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

方法 1.8 (Taylor 近似)

p 次 Taylor 近似は *(**??**)* 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

$$y_{n+1} = y_n + a(t_n, y_n)\Delta_n + \frac{1}{2!}\frac{da}{dt}(t_n, y_n)\Delta_n^2 + \cdots + \frac{1}{p!}\frac{d^{p-1}a}{dt^{p-1}}(t_n)\Delta_n^p \quad (1.13)$$

によって与える.

例 1.9

例えば,2 次 *Taylor* 近似は,

$$y_{n+1} = y_n + a(t_n, y_n)\Delta_n + \frac{1}{2!}\{a_t + a_x a\}\Delta_n^2 \tag{1.14}$$

3 次 *Taylor* 近似は,

$$y_{n+1} = y_n + a(t_n, y_n)\Delta_n + \frac{1}{2!}\{a_t + a_x a\}\Delta_n^2 + \frac{1}{3!}\{a_{tt} + 2a_{tx}a + a_{xx}a^2 + a_t a_x + a_x^2 a \tag{1.15}$$

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.

偏微分を計算するのが面倒

$$a_t(t_n, y_n) \approx \frac{a(t_{n+1}, y_n) - a(t_n, y_n)}{\Delta_n}, a_x(t_n, y_n) \approx \frac{a(t_n, y_{n+1}) - a(t_n, y_n)}{y_{n+1} - y_n} \tag{1.16}$$

もう一度一段法について考えてみる.

方法 1.10 (2 次 Runge-Kutta 法)

2 次 Runge-Kutta 法は

$$y_{n+1} = y_n + \Psi(t_n, y_n, \Delta_n)\Delta_n, \quad y_0 = x_0 \tag{1.17}$$

という一段法の *increment function* に対して,

$$\Psi(t, y, \Delta) = \alpha\, a(t, x) + \beta\, a(t + \gamma\Delta, x + \gamma a(t, x)\Delta) \tag{1.18}$$

を代入することによって得られる.

もう一度一段法について考えてみる.

方法 1.10 (2 次 Runge-Kutta 法)

2 次 Runge-Kutta 法は

$$y_{n+1} = y_n + \Psi(t_n, y_n, \Delta_n)\Delta_n, \quad y_0 = x_0 \tag{1.17}$$

という一段法の *increment function* に対して,

$$\Psi(t, y, \Delta) = \alpha\, a(t, x) + \beta\, a(t + \gamma\Delta, x + \gamma a(t, x)\Delta) \tag{1.18}$$

を代入することによって得られる.

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

方法 1.11 (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

$$y_{n+1} = y_n + \frac{1}{6}\{k_n^{(1)} + 2k_n^{(2)} + 2k_n^{(3)} + k_n^{(4)}\}\Delta_n \tag{1.19}$$

によって与える. ただし,

$$k_n^{(1)} = a(t_n, y_n), \tag{1.20}$$

$$k_n^{(2)} = a\left(t_n + \frac{1}{2}\Delta_n, y_n + \frac{1}{2}k_n^{(1)}\Delta_n\right), \tag{1.21}$$

$$k_n^{(3)} = a\left(t_n + \frac{1}{2}\Delta_n, y_n + \frac{1}{2}k_n^{(2)}\Delta_n\right), \tag{1.22}$$

$$k_n^{(4)} = a\left(t_{n+1}, y_n + k_n^{(3)}\Delta_n\right) \tag{1.23}$$

方法 1.12 (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

$$y_{n+1} = \sum_{j=1}^{k} \alpha_j y_{n+1-j} + \sum_{j=0}^{k} \beta_j a(t_{n+1-j}, y_{n+1-j})\Delta \tag{1.24}$$

によって与える. ただし α_j, β_j は定数である.

例 1.13

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

$$y_{n+1} = y_{n-1} + 2a(t_n, y_n)\Delta \tag{1.25}$$

Milne 法

$$y_{n+1} = y_{n-3} + \frac{4}{3}\{2a(t_n, y_n) - a(t_{n-1}, y_{n-1}) + 2a(t_{n-2}, y_{n-2})\}\Delta \tag{1.26}$$

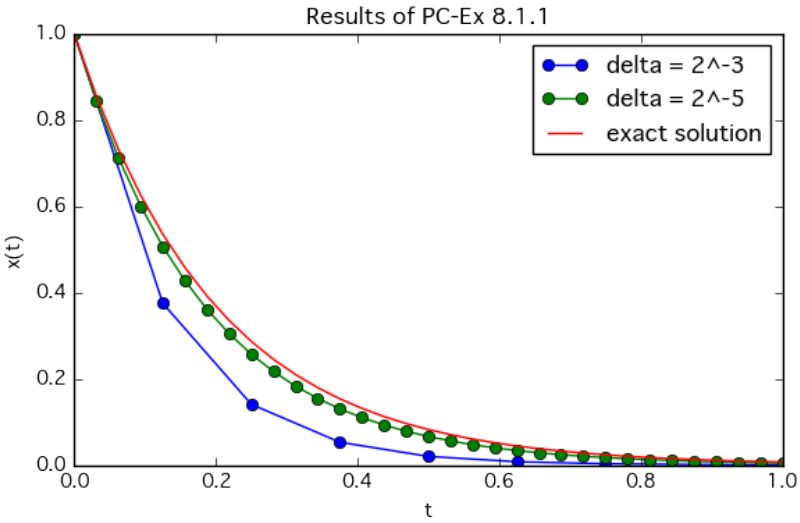
Adams-Moulton 法

$$y_{n+1} = y_n + \frac{1}{12}\{5a(t_{n+1}, y_{n+1}) + 8a(t_n, y_n) - a(t_{n-1}, y_{n-1})\}\Delta \tag{1.27}$$

```python
import matplotlib.pyplot as plt
import math
import numpy as np
import warnings
warnings.simplefilter("ignore")

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]
```

PC-Exercise 8.1.1
Apply the Euler method (1.2) to the VIP $\frac{dx}{dt} = -5x, x(0) = 1$, with $\Delta = 2^{-3}, 2^{-5}$ over $0 \leq t \leq 1$

```
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()
```

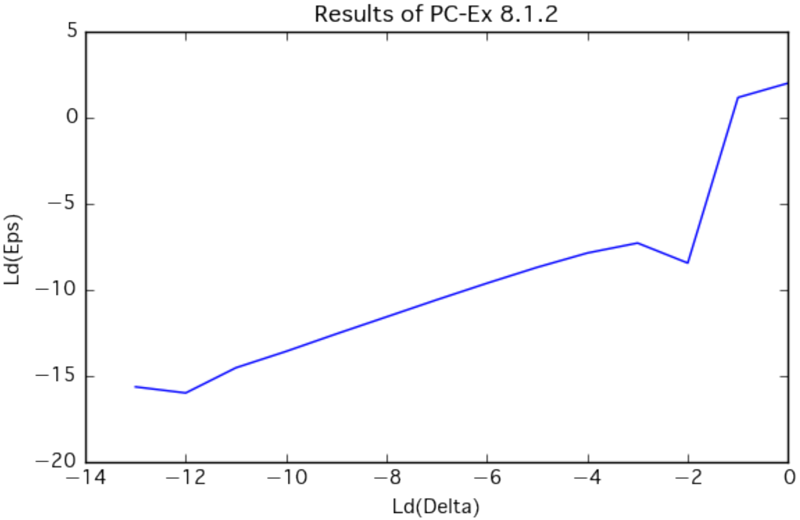

Results of PC-Ex 8.1.1
delta = 2^-3
delta = 2^-5
exact solution
x(t)
t
0.0
0.2
0.4
0.6
0.8
1.0

PC-Exercise 8.1.2

For the IVP in PC-Exercise 8.1.1 calculate the global discretization error at time = 1 for the Euler method with time steps of equal length $\Delta = 1, 2^{-1}, 2^{-2}, \cdots, 2^{-13}$,rounding off to 5 singnificant digits.Plot the logarithm to the base 2 of these error against $\log_2 \Delta$ and determine the slope of the resulting curve.

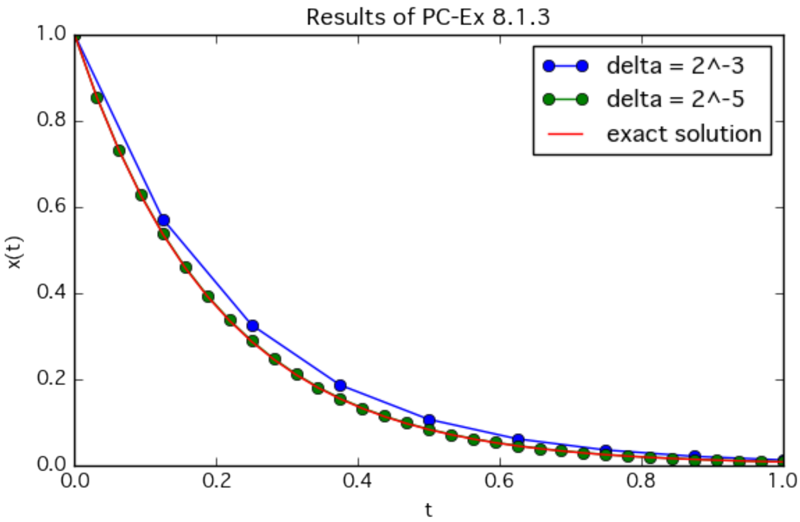
```
def roundoff(x,i=5):
    return float(format(x,'.' + str(i) + 'g'))

def euler_method_round(a,x0,delta):
    t =[roundoff(delta*x) for x in range(int(1/delta)+
    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
```

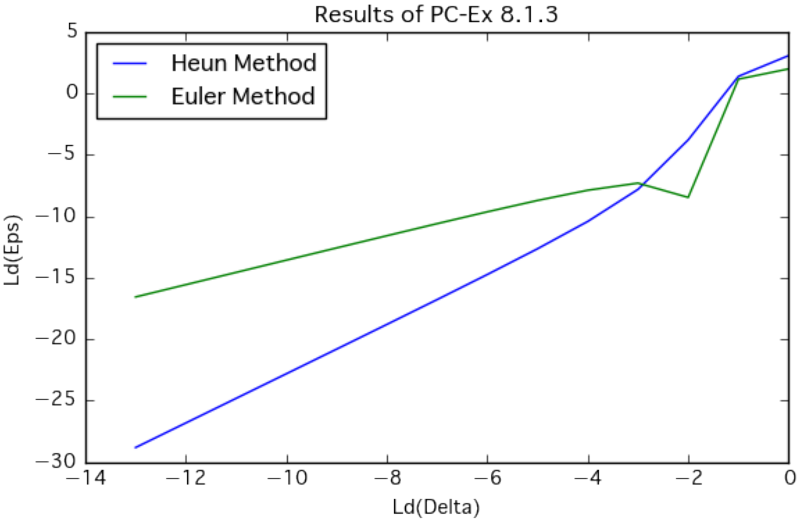
```
from math import fabs,exp,log
error = lambda i : fabs(exp(-5*euler_method_round(a811
                      - euler_method_round(a811,1.0,
y=[log(roundoff(error(i)),2) for i in range(0,-14,-1)]
plt.plot(x,y)
plt.show()
plt.close()
```


Results of PC-Ex 8.1.2
5
0
−5
−10
−15
−20
−14
−12
−10
−8
−6
−4
−2
0
Ld(Eps)
Ld(Delta)

PC-Exercise-8.1.3
Repeat PC-Exercise 8.1.2 with the usual arithmetic of the PC for the modified trapezoidal method (1.12). Compare the results with those for the Euler method.

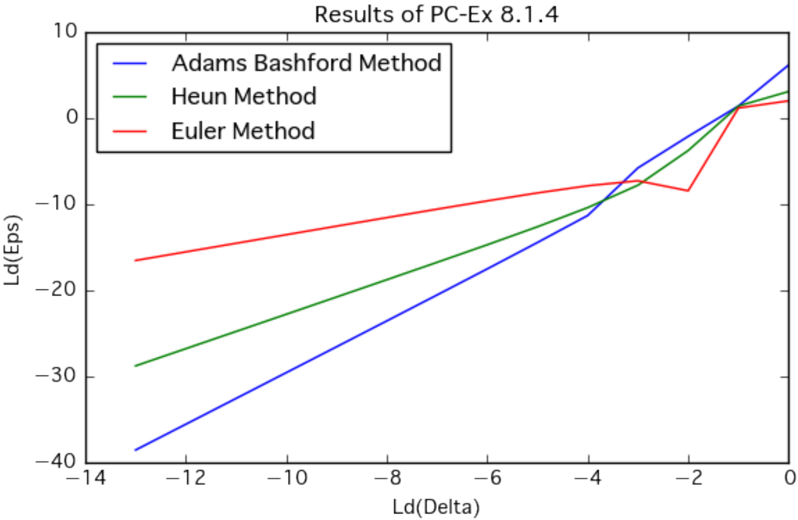
```
def heun_method(a,x0,delta):
    t =[delta*x for x in range(int(1/delta)+1)]
    y =[x0]
    for tn in t[:-1]:
        y_ = y[-1]+a(tn,y[-1])*delta
        y.append(y[-1]+(a(tn,y[-1])+a(tn,y_))/2*delta)
    return [t,y]
```


Results of PC-Ex 8.1.3
delta = 2^-3
delta = 2^-5
exact solution
x(t)
t
0.0
0.2
0.4
0.6
0.8
1.0


Results of PC-Ex 8.1.3
Heun Method
Euler Method
Ld(Eps)
Ld(Delta)
5
0
−5
−10
−15
−20
−25
−30
−14
−12
−10
−8
−6
−4
−2
0

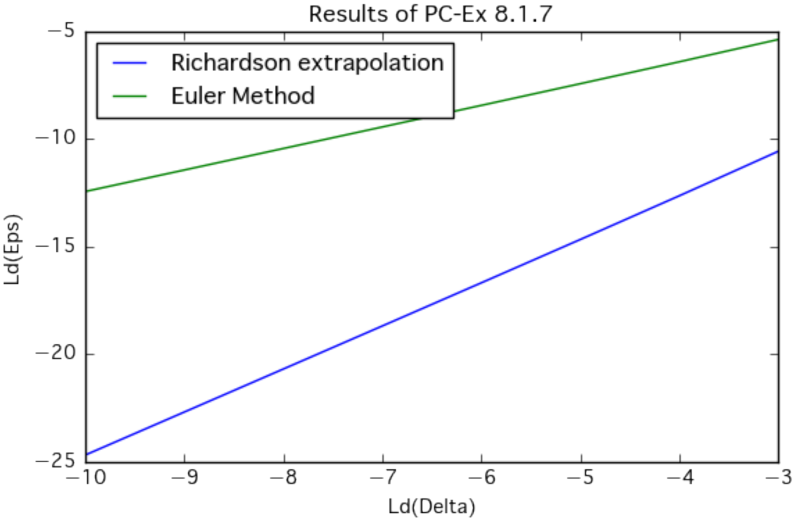
PC-Exercise 8.1.5
Repeat PC-Exercise 8.1.3 using the 3-step Adams-Bashford method (1.14) with the Heun method (1.12) as its starting routine.

```
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-del

    return [t,y]
```


Results of PC-Ex 8.1.4
Adams Bashford Method
Heun Method
Euler Method
Ld(Eps)
Ld(Delta)
10
0
−10
−20
−30
−40
−14
−12
−10
−8
−6
−4
−2
0

PC-Exercise 8.1.7
Compare the error of the Euler and Richardson extrapolation approximations of x(1) for the solution of the initial value problem
$\frac{dx}{dt} = -x, \ x(0) = 1$

```
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
```


Results of PC-Ex 8.1.7
Richardson extrapolation
Euler Method
−5
−10
−15
−20
−25
−10
−9
−8
−7
−6
−5
−4
−3
Ld(Eps)
Ld(Delta)

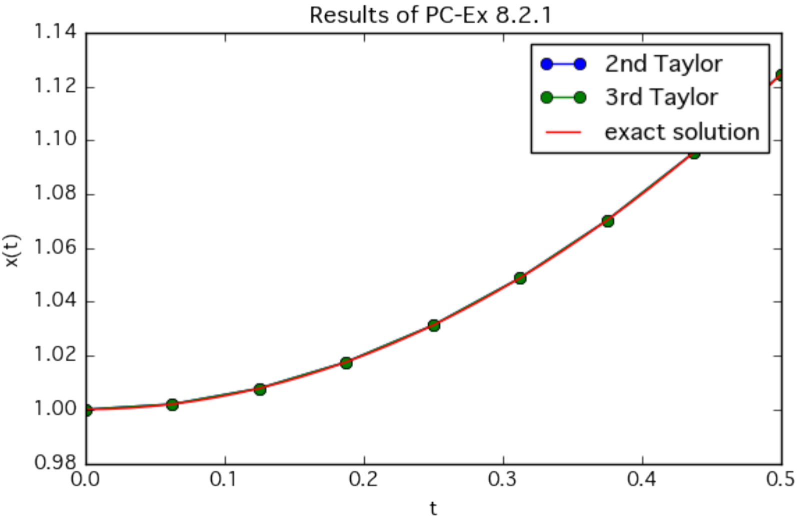
PC-Exercise 8.2.1
Use the 2nd order truncated Taylor method (1.2) with equal length time steps $\Delta = 2^{-3}, 2^{-2}, \cdots, 2^{-10}$ to calculate approximations to the solution $x(t) = 2/(1 + e^{\{\text{-t}}2\})$ of the initial value problem

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

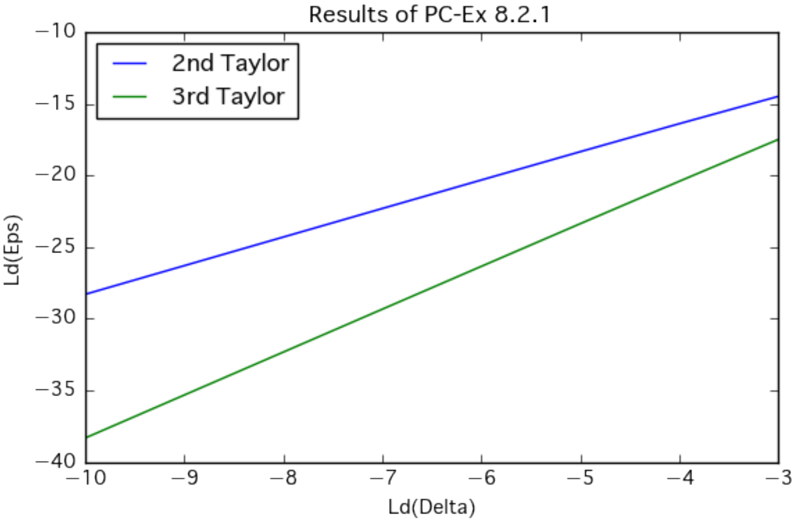
over the interval $0 \leq t \leq 0.5$.

```
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 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[-
    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[-
                (a_tt(tn,y[-1])+2*a_tx(tn,y[-1])*a(tn,
                 + a_x(tn,y[-1])*a_x(tn,y[-1])*a(tn,y[

    return [t,y]
```

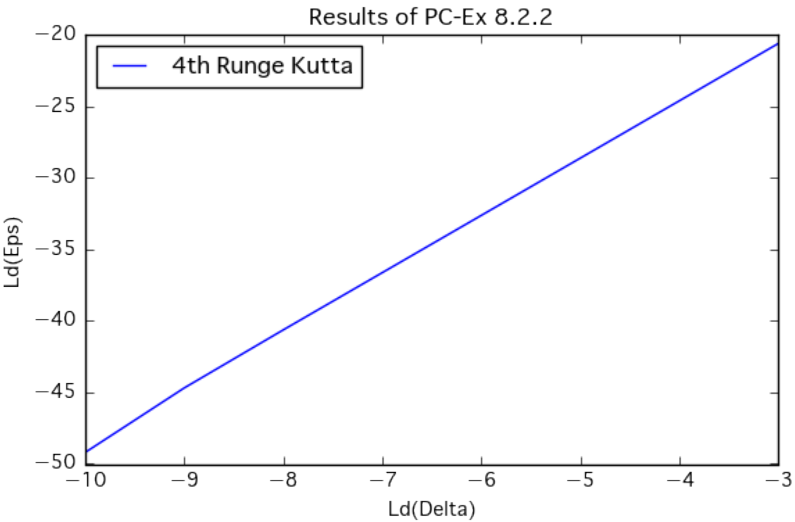


Results of PC-Ex 8.2.1


Results of PC-Ex 8.2.1
2nd Taylor
3rd Taylor
Ld(Eps)
Ld(Delta)
−10
−15
−20
−25
−30
−35
−40
−10
−9
−8
−7
−6
−5
−4
−3

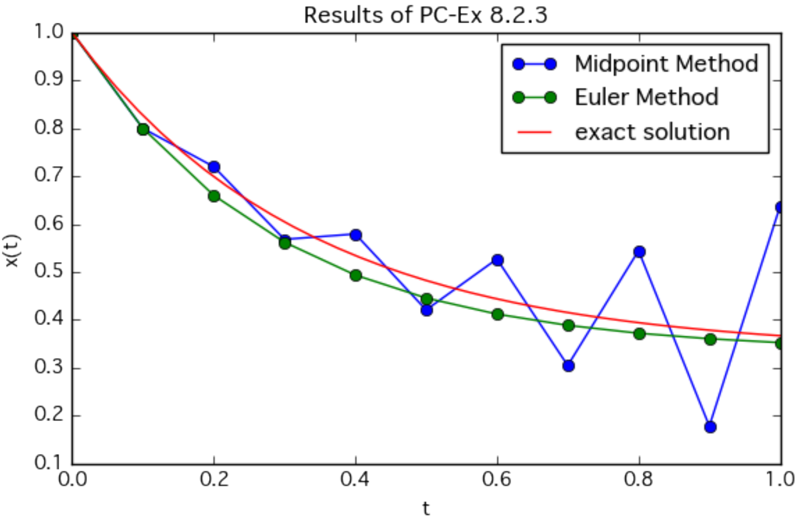
PC-Exercise 8.2.2
Repeat PC-Exercise 8.2.1 using the 4th order Runge-Kutta method(2.8) with equal length time steps $\Delta = 2^{-2}, \cdots, 2^{-7}$

```
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]
```


Results of PC-Ex 8.2.2
4th Runge Kutta
Ld(Eps)
Ld(Delta)
−20
−25
−30
−35
−40
−45
−50
−10
−9
−8
−7
−6
−5
−4
−3

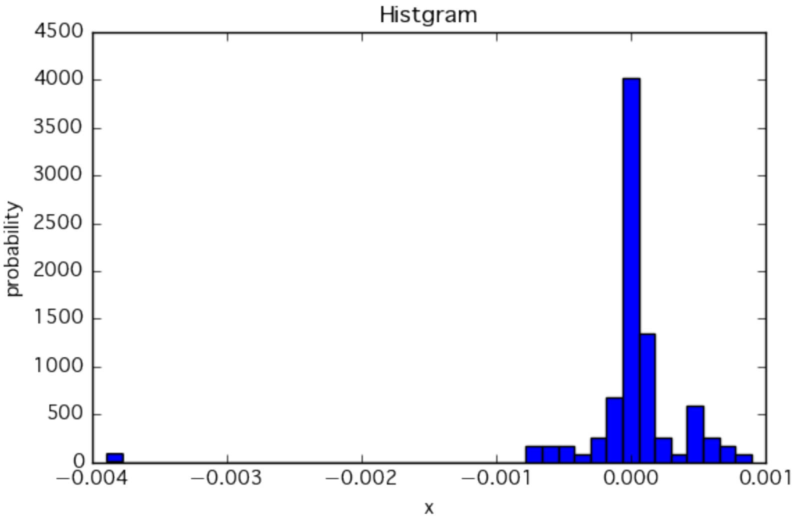
PC-Exercise 8.2.3 Calculate the discretiztion errors in using the Euler method (1.2) and the midpoint method (2.10) started with the Euler method to approximate the solution $x(t) = 2/3e^{-3t} + 1/3$ of the initial value problem
$\frac{dx}{dt} = -3x + 1,\ x(0) = 1$ over the interval $0 \leq t \leq 1$.Use time steps of equal length $\Delta = 0.1$ and plot on x versus t axes

```
In [9]: 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]
```


Results of PC-Ex 8.2.3
Midpoint Method
Euler Method
exact solution
x(t)
t
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.2
0.4
0.6
0.8
1.0

PC-Exercise 8.2.3 Calculate 300 iterates of $y\_\{n+1\} = \frac{\pi}{3} y\_n$ with initial value y_0 = 0.1 using the prescribed arithmetic of the PC, at each step rounding the value of$y_\{n+1\}$ obtained to four siginificant figures. Plot the relative frequencies of the roundoff errors in histogram on the interval $[-5 \times 10^{-5}, 5 \times 10^{-5}]$ using 40 equal subintervals.

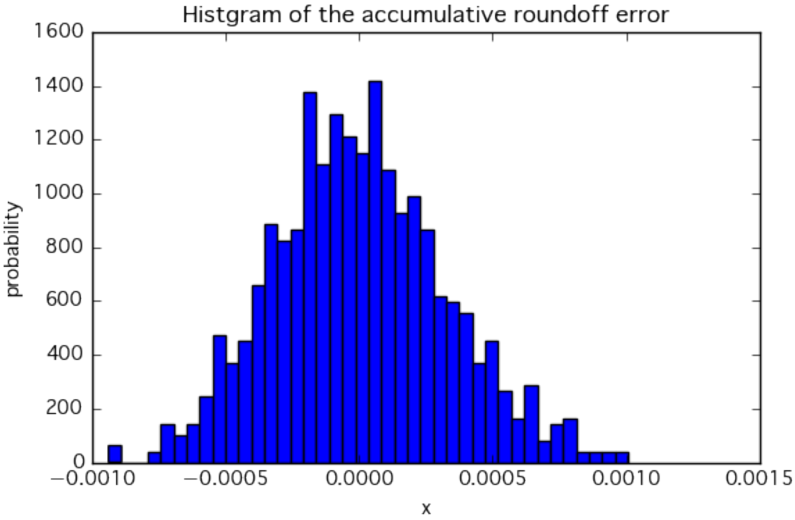
```
#合ってるか分からない

y = [0.1]
e = []
for i in range(100):
    ans = y[-1]*math.pi/3
    ans_ = roundoff(roundoff(y[-1],4)*math.pi/3,4)
    y.append(ans)
    e.append(ans-ans_)

plt.hist(e, normed = True,bins=40)
plt.title("Histgram")
plt.xlabel("x")
plt.ylabel("probability")
plt.show()
```


Histgram
probability
4500
4000
3500
3000
2500
2000
1500
1000
500
0
−0.004
−0.003
−0.002
−0.001
0.000
0.001
x

PC-Exercise 8.4.2
Use the Euler method with equal time steps $\Delta = 2^{-2}$ for the differential equation $\frac{dx}{dt} = x$ over the interval $0 \leq t \leq 1$ with $ N = 10ˆ3$ different initial values $x(0)$ between 0.4 and 0.6. Use both four significant figure arithmetic and the prescribed arithmetic of the PC and determine the final accumulative roundoff error $R_{1/\Delta}$ in each case, plotting them in a histogram on the interval $[-5 \times 10^{-4}, 5 \times 10^{-4}]$ with 40 equal subintervals. In addition, calculate the sample mean and sample variance of the $R_{1/\Delta}$

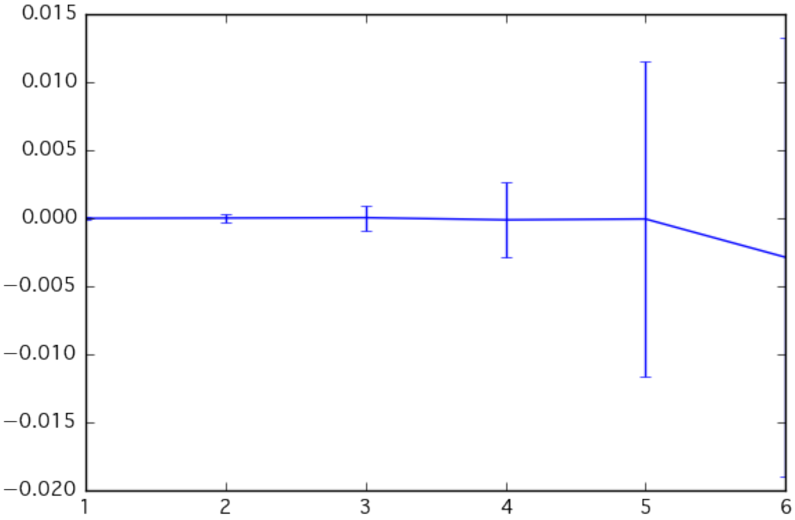
```
x1 = 0.4
x2 = 0.6
t0 = 0
t1 = 1
N = 10**3
delta = 2**(-2)
R = []
x0s = list(np.linspace(x1,x2,N))
a = lambda t,x : x

for x0 in x0s:
    r = 0
    t = list(np.arange(t0,t1+delta,delta))
    y =[x0]
    y_ = [x0]
    for tn in t[:-1]:
        y.append(y[-1]+a(tn,y[-1])*delta)
        y_.append(roundoff(y_[-1],4)+roundoff(a(tn,y_
```

```
for x0 in x0s:
    r = 0
    t = list(np.arange(t0,t1+delta,delta))
    y =[x0]
    y_ = [x0]
    for tn in t[:-1]:
y.append(y[-1]+a(tn,y[-1])*delta)
y_.append(roundoff(y_[-1],4)+roundoff(a(tn,y_[-1])*de
r = r + y[-1] - y_[-1]
    R.append(r)

plt.hist(R, normed = True,bins=40)
 plt.title("Histgram of the accumulative roundoff err
plt.xlabel("x")
plt.ylabel("probability")
plt.show()
print("the sample mean: " + str(np.mean(R)))
```




```
the sample mean: 1.025e-06
```

PC-Exercise 8.4.3 Repeat PC-Exercise 8.4.2 with N = 200 and with time steps $\Delta = 2^{-2}, 2^{-3}, 2^{-4} and,\ 2^{-5}$, determining $R_{1/\Delta}$ in each case. Plot the 90% confidence intervals for the mean value of the error against Δ


0.015
0.010
0.005
0.000
−0.005
−0.010
−0.015
−0.020
1
2
3
4
5
6