Numerical solutions of differential equations

微分方程式の数値解法

Motion of planets – Analytical solution

(惑星の運動 – 解析解)

$$m\frac{d^2\mathbf{r}}{dt^2} = -G\frac{mM}{r^2}\frac{\mathbf{r}}{r} \qquad mr^2\frac{d\theta}{dt} = l$$

l:a constant, conservation of angular momentum

$$\frac{1}{2}m\left(\frac{dr}{dt}\right)^2 + m\left(\frac{l^2}{2m^2r^2} - \frac{GM}{r}\right) = E$$

$$r(\theta) = \frac{b}{1+\varepsilon\cos(\theta-\delta)} \qquad b = \frac{l^2}{mc} \qquad \varepsilon = \sqrt{1+2El^2/mc^2}$$

Elliptic equations **(楕円方程式)**

Long radius of ellipse	$a' = 2b/(1-\varepsilon^2)$
Short radius of ellipse	$b' = 2b/\sqrt{1-\varepsilon^2}$
Eccentricity (離心率) 焦点間の距離/長径	$\varepsilon = \sqrt{1+2El^2/mc^2}$
Close distance point (近点距離)	$q = a'(1\text{-}e) = b/(1+\varepsilon)$
Long distance point (遠点距離)	$Q = a'(1+e) = b/(1\text{-}\varepsilon)$
Period (周期)	$T = 2\pi\sqrt{ma^3/c}$

Normalization of equation

(方程式の規格化/無次元化)

$$m\frac{d^2\mathbf{r}}{dt^2} = -G\frac{mM}{r^2}\frac{\mathbf{r}}{r}$$

Convert variables to T and R by representative constants τ_0 and l_0

$$t = \tau_0 T \qquad r = l_0 R$$

τ_0, l_0: Time and length specific to the system
Choose so that T and R will be of order1

$$m\frac{l_0}{{\tau_0}^2}\frac{d^2\mathbf{R}}{dT^2} = -G\frac{1}{{l_0}^2}\frac{mM}{R^2}\frac{\mathbf{R}}{R}$$

E.g., for planet simulation
- τ_0 = Revolution / Rotation period (公転 / 自転周期)
- l_0 = Revolution radius, Astronomy unit

for molecular dynamics (MD)
- τ_0 = MD time step
- l_0 = Bohr radius (atomic unit)

$$\frac{d^2\mathbf{R}}{dT^2} = -G'\frac{M}{R^2}\frac{\mathbf{R}}{R}$$

$$G' = \frac{G{\tau_0}^2}{{l_0}^3}$$

First-order diff. eq. : Euler formula (オイラー法)

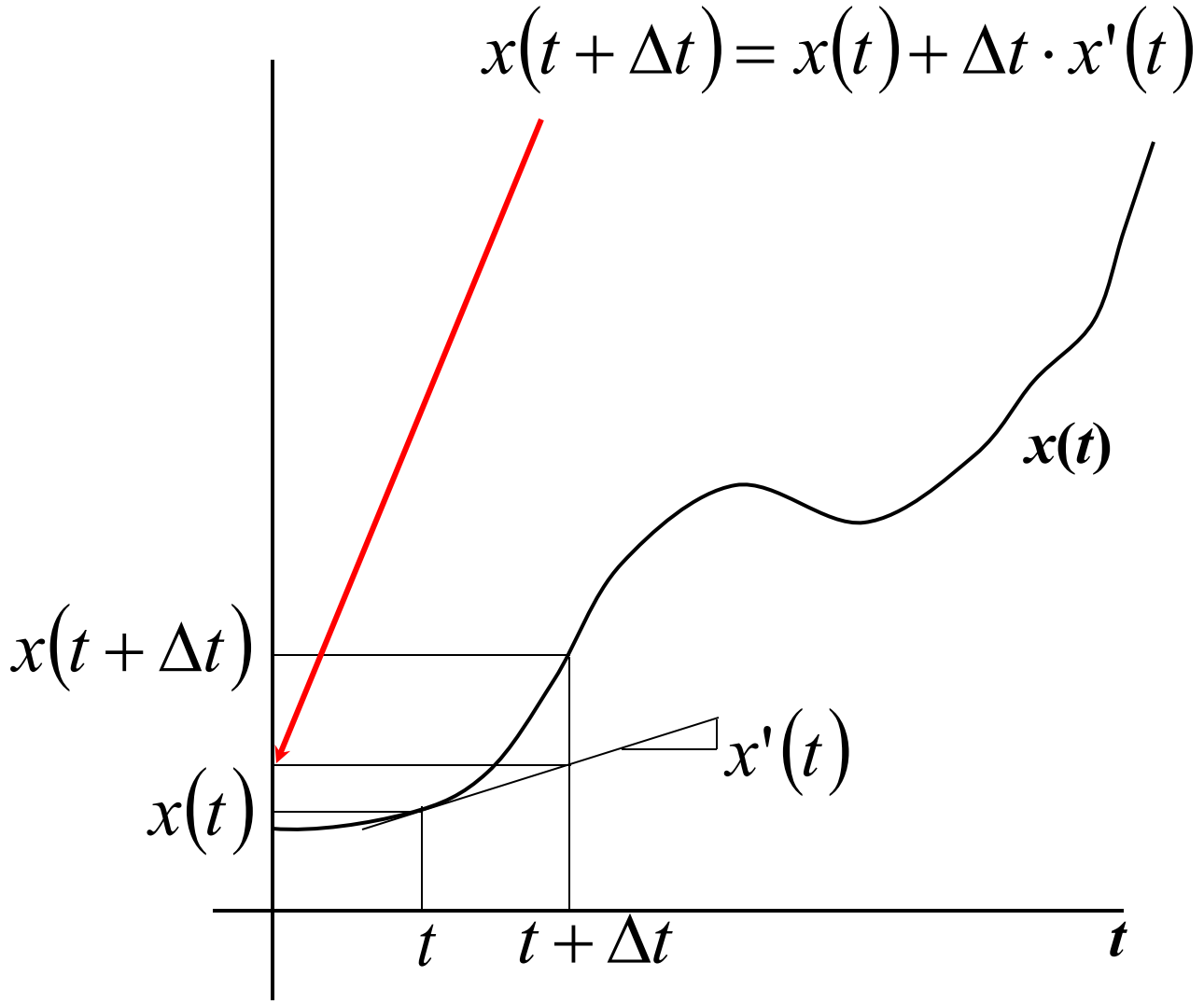
$$\frac{dx}{dt} = f(t, x)$$

$$\frac{x(t + \Delta t) - x(t)}{\Delta t} = f\big(t, x(t)\big)$$

$$x(t + \Delta t) = x(t) + \Delta t \cdot f\big(t, x(t)\big)$$

- **Accuracy not good**
- **Asymmetric with respect to t, $t+\Delta t$**

Illustrative image of Euler method



First-order diff. eq. : Heun formula (ホイン法)

$$\frac{dx}{dt} = f\left(t, x(t)\right)$$

- **Average the Euler formula at t and $t+\Delta t$**

$$x(t+\Delta t) = x(t) + \frac{1}{2}\Delta t[f(t, x(t)) + \boldsymbol{f(t+\Delta t, x(t+\Delta t))}]$$

Problem: $x(t+\Delta t)$ and $f(t+\Delta t, x(t+\Delta t))$ are unknown

=> Use $x(t+\Delta t)$ by Euler formula

$$x(t+\Delta t) \sim x(t) + \Delta t f(t) = x(t) + k_0$$

$$k_0 = \Delta t \cdot f\left(t, x(t)\right)$$

$$k_1 = \Delta t \cdot f\left(t+\Delta t, x\left(t+\Delta t\right)\right) \sim \Delta t \cdot f\left(t+\Delta t, x\left(t\right)+k_0\right)$$

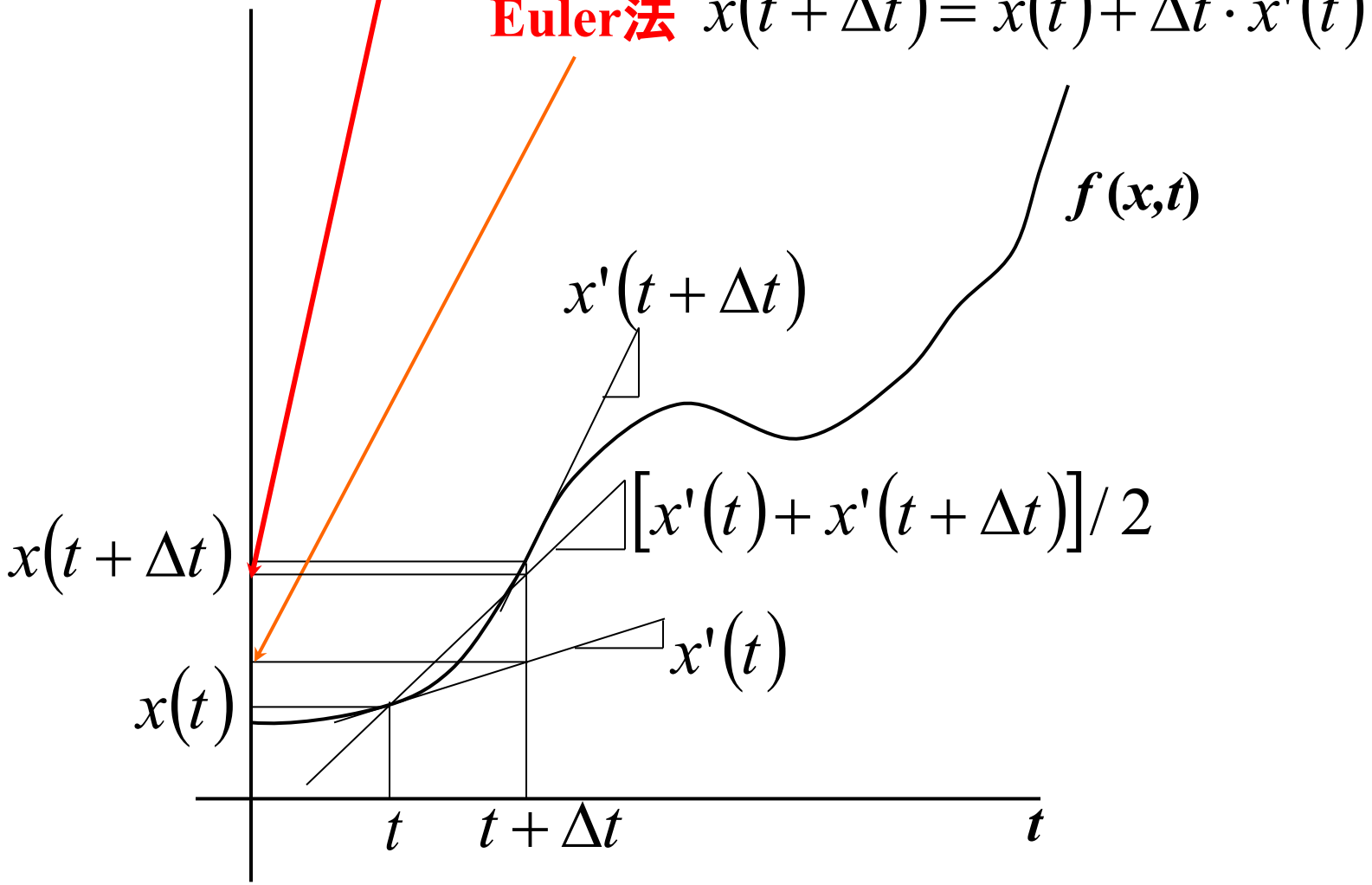
$$x(t+\Delta t) = x(t) + \frac{k_0 + k_1}{2}$$

Illustrative image of Heun method

$$\frac{dx}{dt} = x'(T) = f(t, x(t))$$

Heun法 $x(t+\Delta t) = x(t) + \Delta t \cdot x'_{avg}(t)$

Euler法 $x(t+\Delta t) = x(t) + \Delta t \cdot x'(t)$



First-order differential equation

$$\frac{dx}{dt} = f(t, x)$$

Euler formula: $k_0 = \Delta t \cdot f\big(t, x(t)\big)$

$\boldsymbol{x(t + \Delta t) = x(t) + k_0}$

Heun formula: $k_0 = \Delta t \cdot f\big(t, x(t)\big)$

$k_1 = \Delta t \cdot f(t + \Delta t, x(t) + k_0)$

$\boldsymbol{x(t + \Delta t) = x(t) + \frac{k_0 + k_1}{2}}$

Outline of program

```
dt  = 0.01
t0 = 0.0
x0 = 1.0

# dx/dt = dxdt(t, x)
def dxdt(t, x):
    return -x*x
```

```
# Solve by the Euler formula
def diffeq_euler(diff1func, t0, x0, dt):
    k0 = dt * diff1func(t0, x0)
    x1 = x0 + k0
    return x1

x1 = diffeq_euler(dxdt, t0, x0, dt)
```

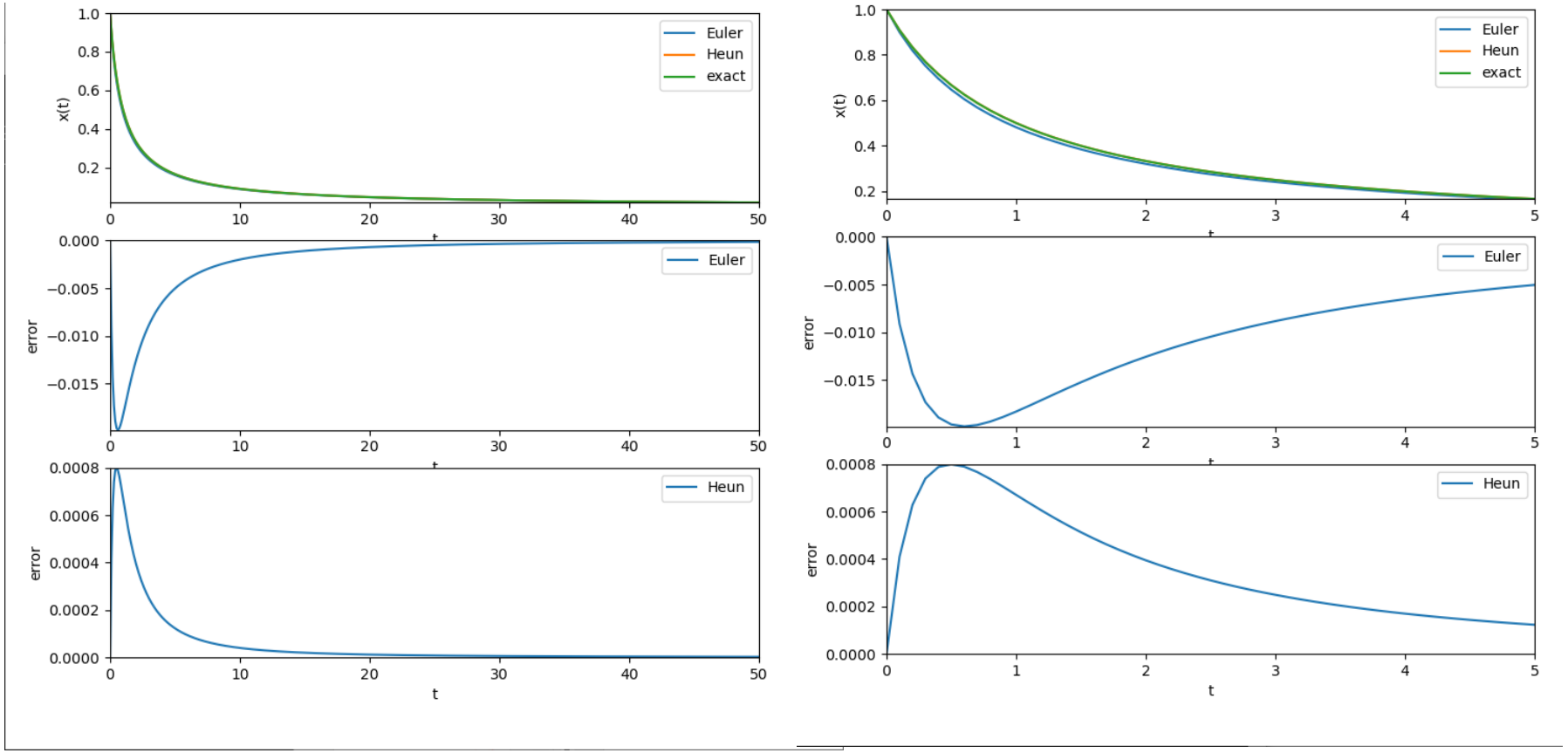
```
# Solve by the Heun formula
def diffeq_heun(diff1func, t0, x0, dt):
    k0 = dt * diff1func(t0, x0)
    k1 = dt * diff1func(t0+dt, x0+k0)
    x1 = x0 + (k0 + k1) / 2.0
    return x1

x1 = diffeq_heun(dxdt, t0, x0, dt)
```

Program: Euler vs. Heun methods

Usage: python diffeq_euler_heun.py x0 dt nt iprint_interval

python diffeq_euler_heun.py

$\frac{dx}{dt} = -x^2$ for $x_0 = 1.0, \Delta t = 0.1, n_t = 501$



First-order diff. eq. : Simpson rule (シンプソン則)

$$\int_{x_0}^{x_2} g(x')dx' \sim \frac{1}{3}h\left[g(x_0)+4g(x_1)+g(x_2)\right]= f(x_2)-f(x_0)$$

Solution of $\frac{df(x)}{dx}=g(x)$ **=>** $\frac{dx}{dt}=f(t,x)$

$$x(t+2\Delta t)=x(t)+\frac{1}{3}\Delta t\left[f(t)+4f(t+\Delta t)+f(t+2\Delta t)\right]$$

Problem: $x(t+\Delta t)$ and $x(t+2\Delta t)$ are unknown

=> Use $x(t+\Delta t)$ by Euler or Heun formula

$$x(t+2\Delta t)=x(t)+\frac{k_0+4k_1+k_2}{3}$$

$$k_0=\Delta t\cdot f(t,x)$$
$$k_1=\Delta t\cdot f(t+\Delta t,x+k_0)$$
$$k_2=\Delta t\cdot f(t+2\Delta t,x+k_0+k_1)$$

Convert Δt to a half

$$x(t+\Delta t)=x(t)+\frac{k_0+4k_1+k_2}{6}$$

$$k_1=\Delta t\cdot f(t+\Delta t/2,x+k_0/2)$$
$$k_2=\Delta t\cdot f(t+\Delta t,x+(k_0+k_1)/2)$$

＝＞ Runge-Kutta formula

First-order diff. eq. : Runge-Kutta formula

(ルンゲークッタ公式)

$$\frac{dx}{dt} = f(t,x)$$

$$x(t+\Delta t) = x(t) + \frac{dx}{dt}\Delta t + \frac{1}{2!}\frac{d^2x}{dt^2}\Delta t^2 + \frac{1}{3!}\frac{d^3x}{dt^3}\Delta t^3 + \cdots$$

$$x(t+\Delta t) = x(t) + \mu_1 k_1 + \mu_2 k_2 + \mu_3 k_3 + \cdots$$

$$k_1 = \Delta t \cdot f(t,x)$$

$$k_2 = \Delta t \cdot f(t+\alpha_1\Delta t, x+\beta_1 k_1)$$

$$k_3 = \Delta t \cdot f(t+\alpha_2\Delta t, x+\beta_2 k_1+\beta_3 k_2)$$

Determine μ_i and k_i so as to get minimum error

Number of k_i n => n-stage formula

Formula of $O(\Delta t^p) = 0$ is called 'order p formula'

3-stage 3-order Runge-Kutta formula

(3段3次のRunge-Kutta公式)

$$x(t+\Delta t) = x(t) + \frac{k_0 + 4k_1 + k_2}{6} + O(h^4)$$

$$k_0 = \Delta t \cdot f(t, x)$$

$$k_1 = \Delta t \cdot f(t + \Delta t/2, x + k_0/2)$$

$$k_2 = \Delta t \cdot f(t + \Delta t, x + 2k_1 - k_0)$$

Different from Simpson rule

$(k_0 + k_1)/2$

Different μ_i and k_i can provide the same accuracy

(同じ精度で違う取り方もできる)

$$k^* = \Delta t \cdot f(t + \Delta t/4, x + \Delta x/4)$$

$$k_0 = \Delta t \cdot f(t, x)$$

$$k_1 = \Delta t \cdot f(t + \Delta t/2, x + k^*/2)$$

$$k_2 = \Delta t \cdot f(t + \Delta t, x + k_1)$$

4-stage 4-order Runge-Kutta formula

(4段4次のRunge-Kutta公式)

$$x(t+\Delta t) = x(t) + \frac{k_0 + 2k_1 + 2k_2 + k_3}{6}$$

$$k_0 = \Delta t \cdot f(t, x)$$
$$k_1 = \Delta t \cdot f(t + \Delta t/2, x + k_0/2)$$
$$k_2 = \Delta t \cdot f(t + \Delta t/2, x + k_1/2)$$
$$k_3 = \Delta t \cdot f(t + \Delta t, x + k_2)$$

First-order differential equation

$$\frac{dx}{dt} = f(t, x)$$

Euler formula:

$$k_0 = \Delta t \cdot f(t, x(t))$$
$$\boldsymbol{x(t + \Delta t) = x(t) + k_0}$$

Heun formula:

$$k_0 = \Delta t \cdot f(x(t), \quad t)$$
$$k_1 = \Delta t \cdot f(x(t) + k_0, t + \Delta t)$$
$$\boldsymbol{x(t + \Delta t) = x(t) + \frac{1}{2}(k_0 + k_1)}$$

Simpson formula:

$$k_0 = \Delta t \cdot f(t, x(t))$$
$$k_1 = \Delta t \cdot f(t + \Delta t/2, x(t) + k_0/2)$$
$$k_2 = \Delta t \cdot f(t + \Delta t, x(t) + (k_0 + k_1)/2)$$
$$\boldsymbol{x(t + \Delta t) = x(t) + \frac{1}{6}(k_0 + 4k_1 + k_2)}$$

3-stage 3-order Runge-Kutta formula:

$$k_0 = \Delta t \cdot f(t, x(t))$$
$$k_1 = \Delta t \cdot f(t + \Delta t/2, x(t) + k_0/2)$$
$$k_2 = \Delta t \cdot f(t + \Delta t, x(t) + 2k_1 - k_0)$$
$$\boldsymbol{x(t + \Delta t) = x(t) + \frac{1}{6}(k_0 + 4k_1 + k_2)}$$

4-stage 4-order Runge-Kutta formula:

$$k_0 = \Delta t \cdot f(t, x(t))$$
$$k_1 = \Delta t \cdot f(t + \Delta t/2, x(t) + k_0/2)$$
$$k_2 = \Delta t \cdot f(t + \Delta t/2, x(t) + k_1/2)$$
$$k_3 = \Delta t \cdot f(t + \Delta t, x(t) + k_2)$$
$$\boldsymbol{x(t + \Delta t) = x(t) + \frac{1}{6}(k_0 + 2k_1 + 2k_2 + k_3)}$$

Second-order diff. eq. (二階微分方程式)

$$\frac{d^2\mathbf{r}_i}{dt^2} = \mathbf{F}_i / m_i$$

- **2nd-order diff eq is divided to two simultaneous 1st-order eqs**

  (二階微分方程式の場合、一階微分方程式に分解するのが良い)

$$\frac{d^2x}{dt^2} = f(t, x, v)$$

$$\frac{dv}{dt} = f(t, x, v) \qquad \frac{dx}{dt} = v$$

Euler formula: $v(t + \Delta t) \sim v(t) + \Delta t \cdot \frac{dv}{dt}$

$$\boldsymbol{v(t + \Delta t) = v(t) + \Delta t \cdot f\big(t, x(t), v(t)\big)}$$

$$\boldsymbol{x(t + \Delta t) = x(t) + \Delta t \cdot v(t)}$$

Second-order diff. eq. : Heun formula

(二階微分方程式の解法: ホイン法)

$$\frac{d^2x}{dt^2} = f(t, x, v)$$

$$\frac{dv}{dt} = f(t, x, v)$$

(1) $k_0 = \Delta t \cdot f\big(t, x(t), v(t)\big)$

(3) $k_1 = \Delta t \cdot f(t + \Delta t, \boldsymbol{x(t) + k_0'}, \boldsymbol{v(t) + k_0})$

(4) $v(t + \Delta t) = v(t) + \frac{1}{2}(k_0 + k_1)$

Each step needs to calculate k_0 and k_1: time-consuming for MD

$$\frac{dx}{dt} = v(t, x, v)$$

(2) $k_0{}' = \Delta t \cdot v(t)$

(5) $k_1{}' = \Delta t \cdot v(t + \Delta t)$

(6) $x(t + \Delta t) = x(t) + \frac{1}{2}(k_0{}' + k_1{}')$

Second-order diff. eq. : position Verlet formula

(二階微分方程式の解法: 位置ベルレ法)

$$\frac{d^2x}{dt^2} = f(t, x, v)$$

$$f(x, v, t) = \frac{d^2x(t)}{dt^2} \sim \frac{x(t+\Delta t) - 2x(t) + x(t-\Delta t)}{\Delta t^2}$$

$$x(t+\Delta t) = 2x(t) - x(t-\Delta t) + \Delta t^2 f\big(\boldsymbol{t, x(t), v(t)}\big)$$

$$v(t) = \frac{1}{2\Delta t}\{x(t+\Delta t) - x(t-\Delta t)\}$$

Each step needs to calculate only one $\boldsymbol{f\big(t, x(t), v(t)\big)}$

- **Better accuracy than Euler formula, equivalent to Heun formula**
- **Directly solves a second-order differential equation**
- **Drawback:**
  The subtraction of similar values, $x(t+n\Delta t)$, may cause roundoff error.

velocity Verlet formula

$$\frac{d^2x}{dt^2} = f(t,x)$$

$$\frac{d^2x(t+\Delta t)}{dt^2} \sim \frac{x(t+2\Delta t) - 2x(t+\Delta t) + x(t)}{\Delta t^2}$$

$$x(t+2\Delta t) = 2x(t+\Delta t) - x(t) + \Delta t^2 f(t+\Delta t, x(t+\Delta t))$$

$$x(t+\Delta t) = x(t) + v(t)\Delta t + \frac{1}{2} f\big(t, x(t), v(t)\big)\Delta t^2$$

$$v(t+\Delta t) = v(t) + \frac{\Delta t}{2}\big\{f\big(t, x(t)\big) + f\big(t+\Delta t, x(t+\Delta t)\big)\big\}$$

- **Positions and velocities are synchronized at each time step**.
- **More convenient than position Verlet for evaluating velocities and kinetic energy**.

Program: diffeq2nd_verlet.py

Usage: python diffeq2nd_verlet.py

t	x(cal)	x(exact)	v(cal)
t= 0.00	0.000000	0.000000	1.000000
t= 0.01	0.010000	0.010000	0.999950
t= 0.20	0.198673	0.198669	0.980066
t= 0.40	0.389425	0.389418	0.921060
t= 0.60	0.564652	0.564642	0.825334
t= 0.80	0.717367	0.717356	0.696704
t= 1.00	0.841484	0.841471	0.540299
t= 1.20	0.932053	0.932039	0.362353
t= 1.40	0.985463	0.985450	0.169961
t= 1.60	0.999586	0.999574	-0.029206
t= 1.80	0.973858	0.973848	-0.227209
t= 2.00	0.909305	0.909297	-0.416154
t= 2.20	0.808501	0.808496	-0.588509
t= 2.40	0.675464	0.675463	-0.737400
t= 2.60	0.515499	0.515501	-0.856894
t= 2.80	0.334981	0.334988	-0.942226
t= 3.00	0.141109	0.141120	-0.989994
t= 3.20	-0.058388	-0.058374	-0.998294
t= 3.40	-0.255558	-0.255541	-0.966795
t= 3.60	-0.442539	-0.442520	-0.896752
t= 3.80	-0.611878	-0.611858	-0.790958
t= 4.00	-0.756823	-0.756802	-0.653631
t= 4.20	-0.871595	-0.871576	-0.490246
t= 4.40	-0.951620	-0.951602	-0.307315
t= 4.60	-0.993706	-0.993691	-0.112133

Second-order diff. eq. : Leap-flog formula

(二階微分方程式の解法: かえる跳び法)

Essentially the same as the Verlet formula.
However, Verlet formula

$$x(t+\Delta t) = 2x(t) - x(t-\Delta t) + \Delta t^2 f\big(t, x(t), v(t)\big)$$

includes the subtraction of
***x*(*t*) terms and may cause round-off error.**

Converting the equation to

$$v(t+\Delta t) = v(t-\Delta t) + 2\Delta t \cdot f\big(t, x(t), v(t)\big)$$

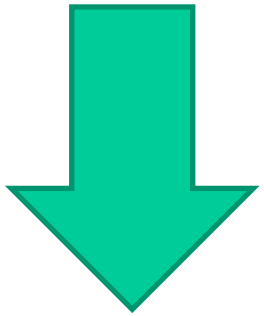
$$x(t+2\Delta t) = x(t) + 2\Delta t \cdot v(t+\Delta t)$$

Can reduce the round-off errors.
Note: Time *t* calculated for *x*(*t*) and *v*(*t*) offset by Δ*t* relative to each other.

Leap Flog vs. Verlet

Confirm the Leap Flog formula is identical to the Verlet formula

Leap Flog $x(t+2\Delta t)=x(t)+2\Delta t\cdot v(t+\Delta t)$



$$v(t-\Delta t)=\frac{x(t)-x(t-\Delta t)}{\Delta t}$$

$$\frac{x(t+\Delta t)-x(t)}{\Delta t}=\frac{x(t)-x(t-\Delta t)}{\Delta t}+2\Delta t\cdot f\big(t,x(t),v(t)\big)$$

$$x(t+\Delta t)=2x(t)-x(t-\Delta t)+2\Delta t\cdot f\big(t,x(t),v(t)\big)$$

Verlet formula

Program: diffeq2nd_2d_euler.py

Two-dimensional second-order differential equation

$$\frac{d^2x}{dt^2} = -x$$

$$\frac{d^2y}{dt^2} = -y$$

Usage: python diffeq2nd_2d_euler.py

t	x(cal)	x(exact)	y(cal)	y(exact)
t= 0.00	0.000000	0.000000	2.000000	2.000000
t= 0.01	0.010000	0.010000	2.000000	1.999900
t= 0.20	0.198862	0.198669	1.962097	1.960133
t= 0.40	0.390186	0.389418	1.845820	1.842122
t= 0.60	0.566322	0.564642	1.655653	1.650671
t= 0.80	0.720212	0.717356	1.399036	1.393413
t= 1.00	0.845671	0.841471	1.086077	1.080605
t= 1.20	0.937633	0.932039	0.729152	0.724716
t= 1.40	0.992364	0.985450	0.342415	0.339934
t= 1.60	1.007603	0.999574	-0.058761	-0.058399
t= 1.80	0.982665	0.973848	-0.458394	-0.454404
t= 2.00	0.918464	0.909297	-0.840535	-0.832294
t= 2.20	0.817482	0.808496	-1.189900	-1.177002
t= 2.40	0.683677	0.675463	-1.492481	-1.474787
t= 2.60	0.522322	0.515501	-1.736110	-1.713778
t= 2.80	0.339800	0.334988	-1.910948	-1.884445
t= 3.00	0.143353	0.141120	-2.009878	-1.979985
t= 3.20	-0.059207	-0.058374	-2.028803	-1.996590
t= 3.40	-0.259811	-0.255541	-1.966806	-1.933596
t= 3.60	-0.450448	-0.442520	-1.826199	-1.793517
t= 3.80	-0.623492	-0.611858	-1.612436	-1.581935
t= 4.00	-0.772001	-0.756802	-1.333901	-1.307287
t= 4.20	-0.890001	-0.871576	-1.001578	-0.980522
t= 4.40	-0.972722	-0.951602	-0.628623	-0.614666
t= 4.60	-1.016792	-0.993691	-0.229835	-0.224305

Program: diffeq2nd_2d_verlet.py

Two-dimensional second-order differential equation

$$\frac{d^2x}{dx^2} = -x$$

$$\frac{d^2y}{dx^2} = -y$$

Usage: python diffeq2nd_2d_verlet.py

t	x(cal)	x(exact)	y(cal)	y(exact)
t= 0.00	0.000000	0.000000	2.000000	2.000000
t= 0.01	0.010050	0.010000	1.999950	1.999900
t= 0.20	0.199666	0.198669	1.961126	1.960133
t= 0.40	0.391372	0.389418	1.844068	1.842122
t= 0.60	0.567475	0.564642	1.653492	1.650671
t= 0.80	0.720954	0.717356	1.396995	1.393413
t= 1.00	0.845691	0.841471	1.084805	1.080605
t= 1.20	0.936713	0.932039	0.729366	0.724716
t= 1.40	0.990390	0.985450	0.344850	0.339934
t= 1.60	1.004584	0.999574	-0.053414	-0.058399
t= 1.80	0.978727	0.973848	-0.449550	-0.454404
t= 2.00	0.913852	0.909297	-0.827762	-0.832294
t= 2.20	0.812544	0.808496	-1.172975	-1.177002
t= 2.40	0.678842	0.675463	-1.471424	-1.474787
t= 2.60	0.518076	0.515501	-1.711211	-1.713778
t= 2.80	0.336656	0.334988	-1.882778	-1.884445
t= 3.00	0.141815	0.141120	-1.979283	-1.979985
t= 3.20	-0.058680	-0.058374	-1.996880	-1.996590
t= 3.40	-0.256836	-0.255541	-1.934867	-1.933596
t= 3.60	-0.444752	-0.442520	-1.795716	-1.793517
t= 3.80	-0.614937	-0.611858	-1.584975	-1.581935
t= 4.00	-0.760607	-0.756802	-1.311046	-1.307287
t= 4.20	-0.875953	-0.871576	-0.984849	-0.980522
t= 4.40	-0.956378	-0.951602	-0.619389	-0.614666
t= 4.60	-0.998674	-0.993691	-0.229235	-0.224305

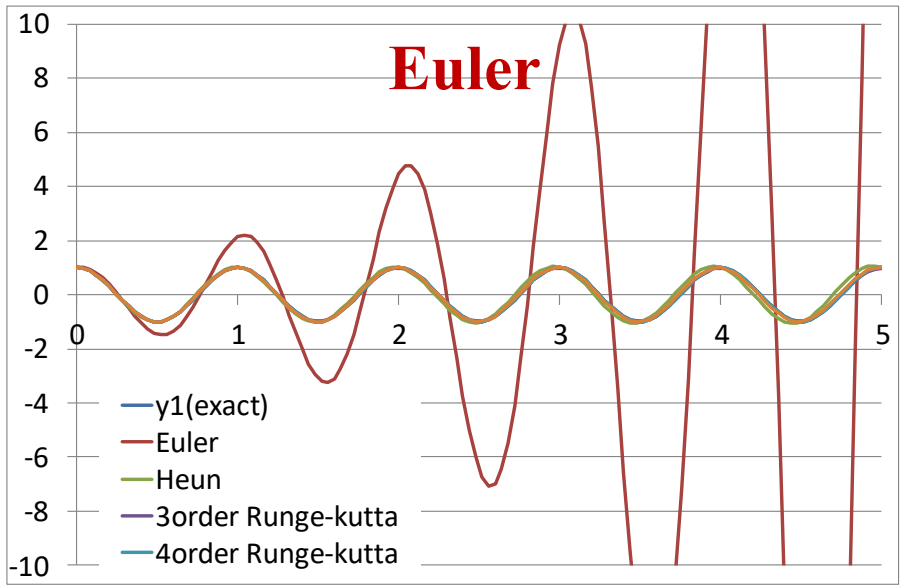
Accuracy of numerical solutions: Diff. eq.

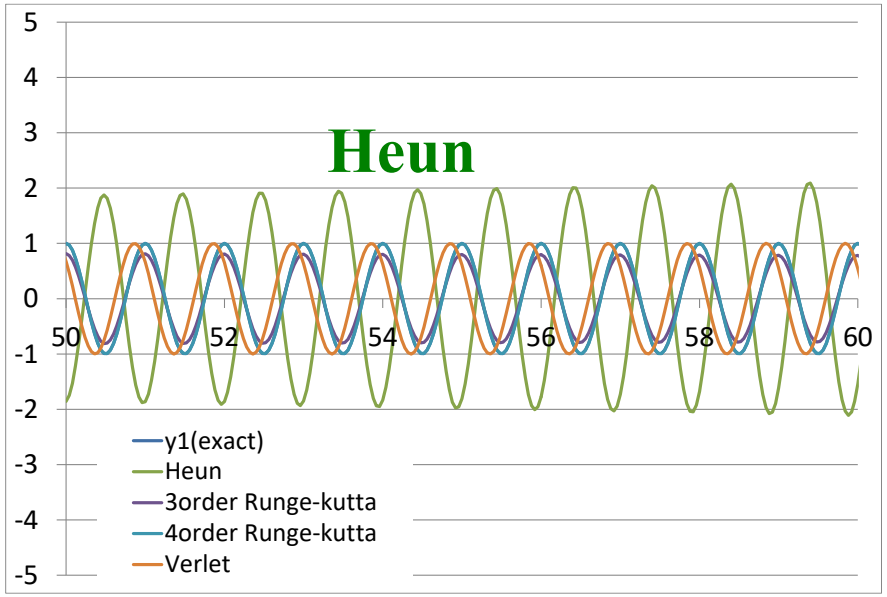
$$\frac{d^2x}{dt^2} = -4\pi^2 x \qquad \left(\frac{dx}{dt} = v, \ \frac{dv}{dt} = -4\pi^2 x \right)$$

Exact ($t = 0$: $x = 1.0$, $v = 0.0$)

$$x = \cos(2\pi t) \quad v = -2\pi \sin(2\pi t)$$

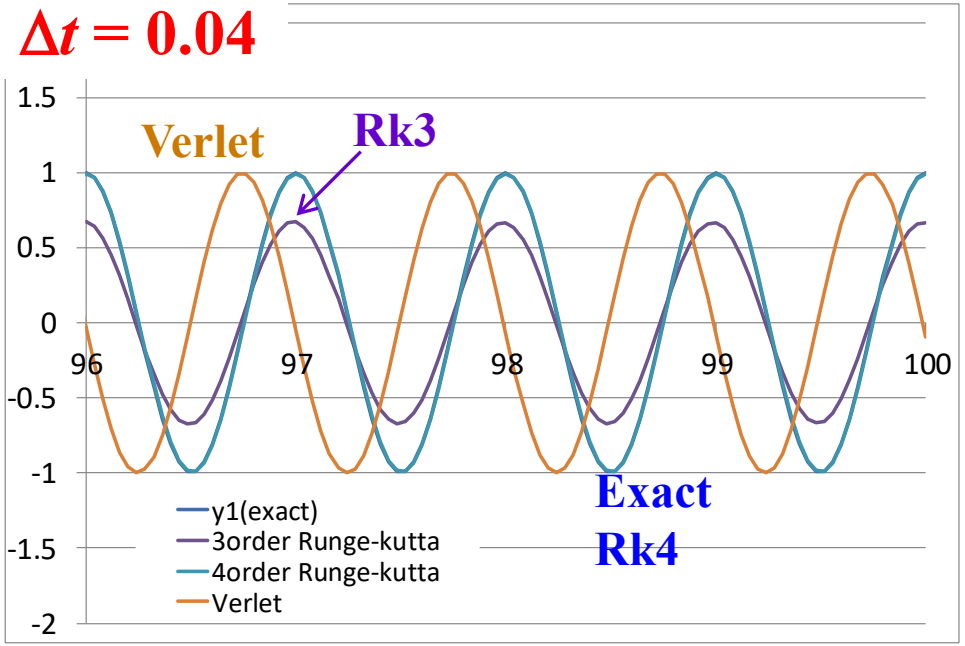
$\Delta t = 0.04$

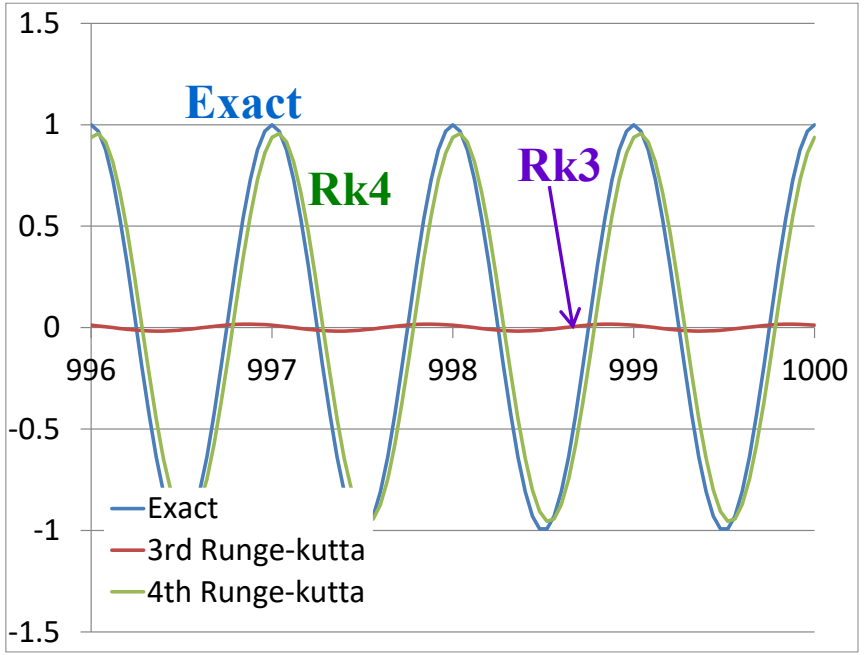




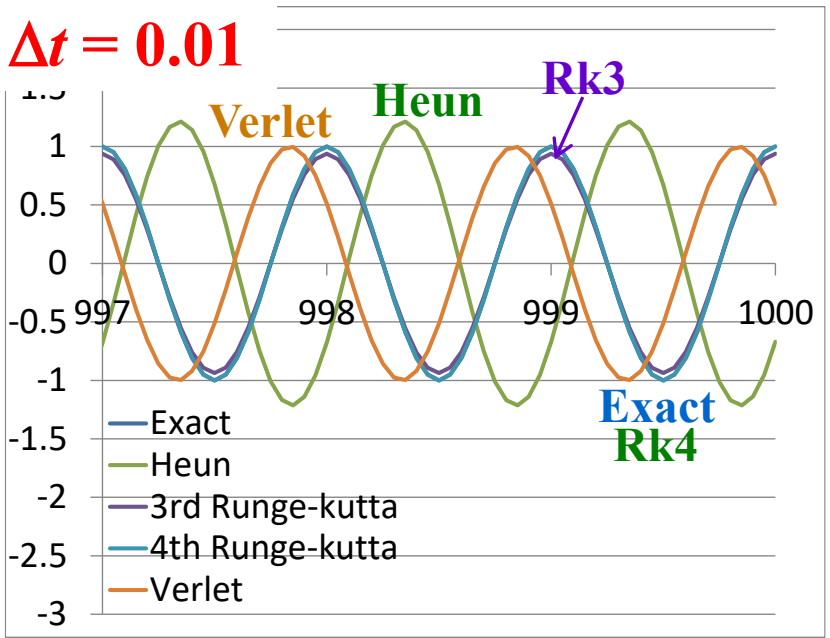
Accuracy of numerical solutions: Diff. eq.

$\Delta t = 0.04$



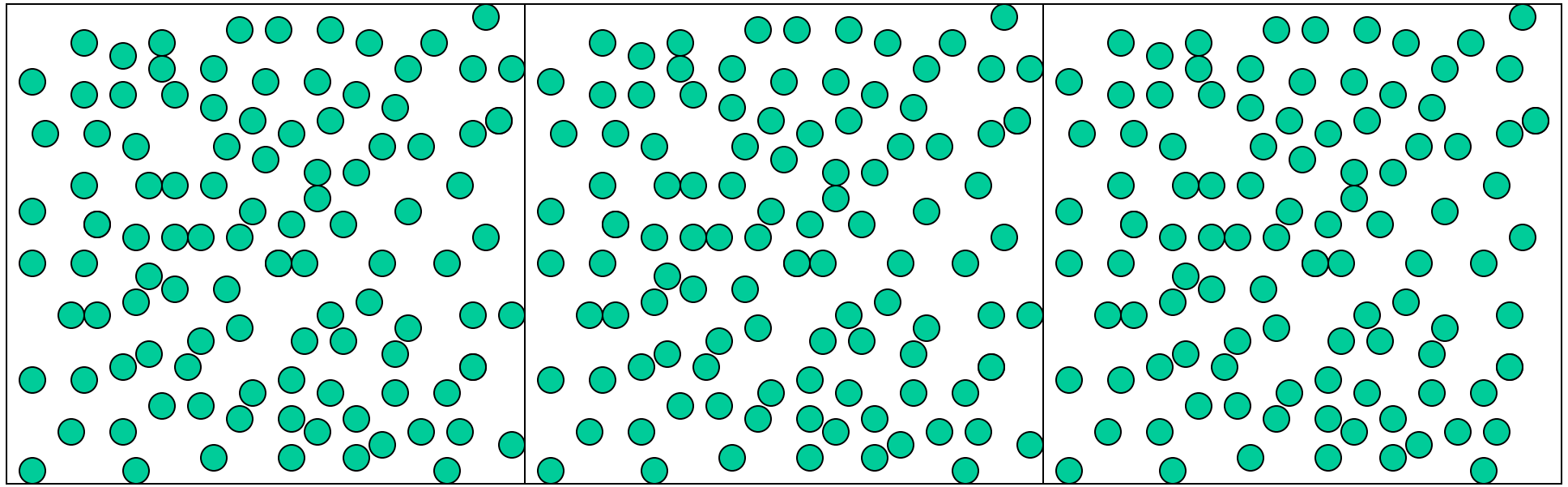


$\Delta t = 0.01$



Molecular dynamics (MD) (分子動力学法)

3D periodic condition: MD cell



$$\mathbf{F}_i = m_i \frac{d^2\mathbf{r}_i}{dt^2}$$

$$\mathbf{v}_i(t+\Delta t) = \mathbf{v}_i(t) + \Delta t \frac{\mathbf{F}_i}{m_i}$$

$$\mathbf{r}_i(t+\Delta t) = \mathbf{r}_i(t) + \Delta t \cdot \mathbf{v}_i(t)$$

Empirical interatomic potential

(経験的原子間ポテンシャル)

Hard core potential

ハードコア（剛体）ポテンシャル

$$\phi(r) = \infty \qquad r \le \sigma$$
$$= 0 \qquad r > \sigma$$

Lennard-Jones (LJ) potential

レナードージョーンズポテンシャル

$$\phi_{ij}(r) = 4\varepsilon_{ij}\left\{\left(\frac{\sigma_{ij}}{r}\right)^{12} - \left(\frac{\sigma_{ij}}{r}\right)^{6}\right\}$$

Born-Mayer-Huggins (BMH) potential

ボルンーメイヤーハギンズ

$$\phi_{ij}(r) = \frac{z_i z_j e^2}{r} + A_{ij} b \cdot \exp\left(\frac{\sigma_i + \sigma_j - r}{\rho}\right) - \frac{C_{ij}}{r^6} - \frac{D_{ij}}{r^8}$$

Kawamura potential（MXDOrto/MXDTricl）

河村ポテンシャル

$$\phi_{ij} = \frac{z_i z_j}{r_{ij}} + f_0\left(b_i + b_j\right)\exp\left(\frac{a_i + a_j - r_{ij}}{b_i + b_j}\right) + \frac{c_i c_j}{r_{ij}^{\,6}}$$

$$\phi_{ij}(r) = \frac{z_i z_j e^2}{r} + f_0\left(b_i + b_j\right)\exp\left(\frac{a_i + a_j - r}{b_i + b_j}\right)$$
$$+ D_{ij}\left(\exp\left[-2\beta_{ij}\left(r - r^*\right)\right] - 2\exp\left[-\beta_{ij}\left(r - r^*\right)\right]\right)$$

Morse potential

Empirical interatomic potential

$$U_{ij}(r_{ij}) = \frac{z_i z_j e^2}{4\pi\varepsilon_0}\frac{1}{r_{ij}} + f_0(b_i + b_j)\exp\left[\frac{a_i + a_j - r_{ij}}{b_i + b_j}\right] + \frac{c_i c_j}{r_{ij}^6}$$

Coulomb potential **Repulsion term** **Dispersion (London interaction)**

Example of Parameters for an ion

Ion charge	**: z_i**	**Fixed to ion formal charge**
~Ion radius	**: a_i**	**Adjust to crystal structure**
~Ion hardness:	**b_i**	**Adjust to elastic constant**
Dispersion	**: c_i**	**Fixed**

Potentials and forces for the ion *i* at r_i

$$U_i(\boldsymbol{r_i}, t) = \sum_j U_{ij}\big(\boldsymbol{r_j}(t) - \boldsymbol{r_i}(t)\big),\ \boldsymbol{F_i}(\boldsymbol{r_i}, t) = -\sum_j \frac{\partial}{\partial \boldsymbol{r_i}} U_{ij}\big(\boldsymbol{r_j}(t) - \boldsymbol{r_i}(t)\big)$$

Most time-consuming term

Better to re-use previous steps,

$\boldsymbol{F_i(r_i, t - \Delta t), F_i(r_i, t - 2\Delta t)}$ **etc**

=> Verlet formula is better than Heun and Runge-Kutta formula

Requirements of algorithms used for MD

Requirements

- Enough accuracy (can be checked by energy / momentum conservation laws)
  symplecticity: avoid accumulation of total energy errors
- Fast calculations (note the most time-consuming process
  is the force calculations, better to re-use the previous results)

Runge-Kutta formula: not suitable for MD

High accuracy, but high cost

It cannot re-use the previous results: Each step requires 3-4 new force evaluations

No symplecticity

Frequently used formula:

- Verlet formula (Leap Flog formula)
- Beeman formula
- Predictor-Corrector method (予測子－修正子法)
  Rahman predictor-corrector method (ラーマンの予測子－修正子法)
  Gear predictor-corrector method (ギアの予測子－修正子法)

What is Smecticity

Hamilton's equations of motion are given by

$$\frac{dq}{dt} = \frac{\partial H}{\partial p}, \quad \frac{dp}{dt} = -\frac{\partial H}{\partial q}$$

Introducing $z = \begin{pmatrix} q(t) \\ p(t) \end{pmatrix}$, they can be written as

$$\frac{dz}{dt} = J\nabla H(z) \qquad \left(J = \begin{pmatrix} 0 & I \\ -I & 0 \end{pmatrix}\right)$$

When the numerical time evolution in MD is expressed as $\mathbf{z}_{n+1} = \Phi_h(\mathbf{z}_n)$ and if $(D\Phi_h)^{\mathsf{T}} J D\Phi_h = J \quad \left(D\Phi_h = \frac{\partial \mathbf{z}_{n+1}}{\partial \mathbf{z}_n}\right)$,

the numerical time evolution preserves the geometric structure of Hamiltonian mechanics.

As a result, energy errors are less likely to accumulate in one direction, making long-time simulations more stable.

シンプレクティック性

Hamilton の運動方程式は、$\frac{dq}{dt} = \frac{\partial H}{\partial p}$、$\frac{dp}{dt} = -\frac{\partial H}{\partial q}$ で与えられる。

$z = \begin{pmatrix} q(t) \\ p(t) \end{pmatrix}$ と書くと、$\frac{dz}{dt} = J\nabla H(z)$ $\left(J = \begin{pmatrix} 0 & I \\ -I & 0 \end{pmatrix}\right)$と書ける。

MDの数値時間変化を $\mathbf{z}_{n+1} = \Phi_h(\mathbf{z}_n)$ と表すとき、

$$(D\Phi_h)^{\mathsf{T}} J D\Phi_h = J \quad \left(D\Phi_h = \frac{\partial \mathbf{z}_{n+1}}{\partial \mathbf{z}_n}\right)$$

であれば、数値時間発展はHamilton力学の幾何学的構造を保持する。

その結果、エネルギー誤差が一方向に蓄積しにくく、

長時間計算でも安定しやすい。

Suitability for MD

Algorithm 方法	Error order/Accuracy 誤差/精度	# of force evaluations / step 力評価回数	Symplecticity シンプレクティック性	Suitability for MD MD適性
Euler	Order 1/Low 1次/低い	1	No	Poor
Heun	Order 2/Middle 2次/中程度	2	No	Better, but still poor
RK4	Order 4/High 4次/高い	4	No	Good accuracy, but poor long-term stability
Verlet / leap-frog	Order 2 2次	1*	Yes	Good
velocity Verlet	Order 2 2次	1*	Yes	Excellent; standard in MD

* One new force evaluation per step after initialization.

Program: Planet simulation

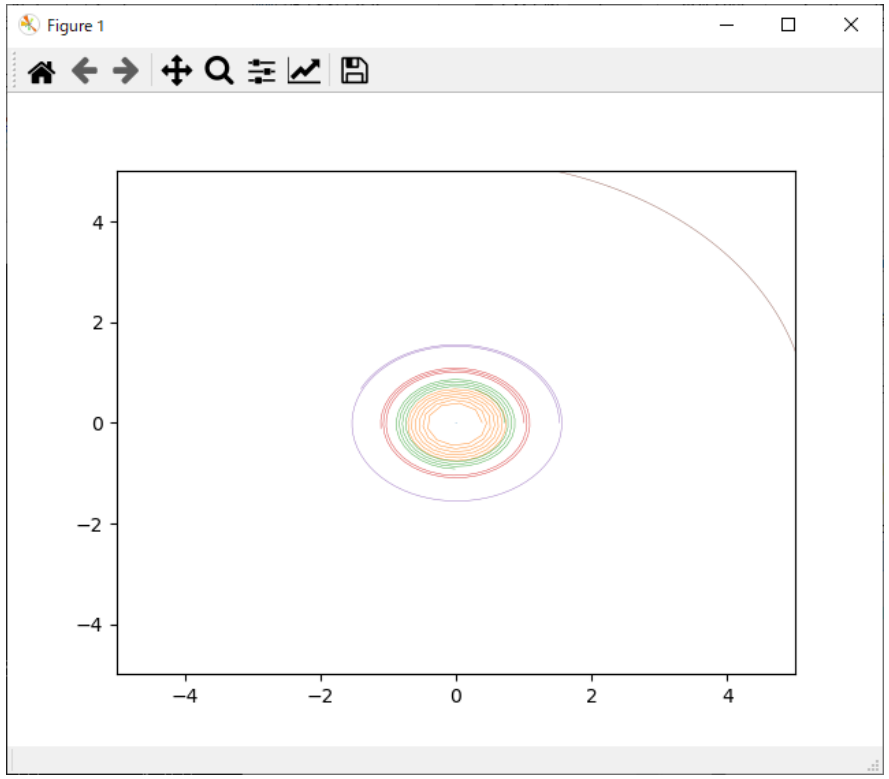
Usage: python diffeq2nd_planet.py solver dt nt

solver: ‘Euler’ or ‘Verlet’

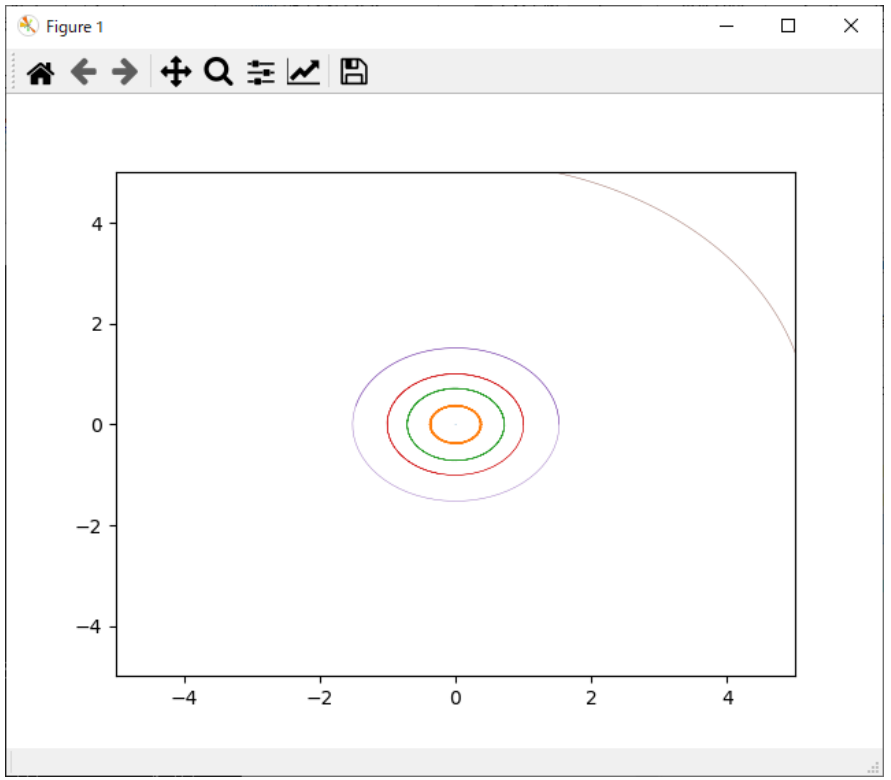
dt: time step in day (time is normalized by a day)

nt: number of steps

python diffeq2nd_planet.py **Euler** 0.2 5000

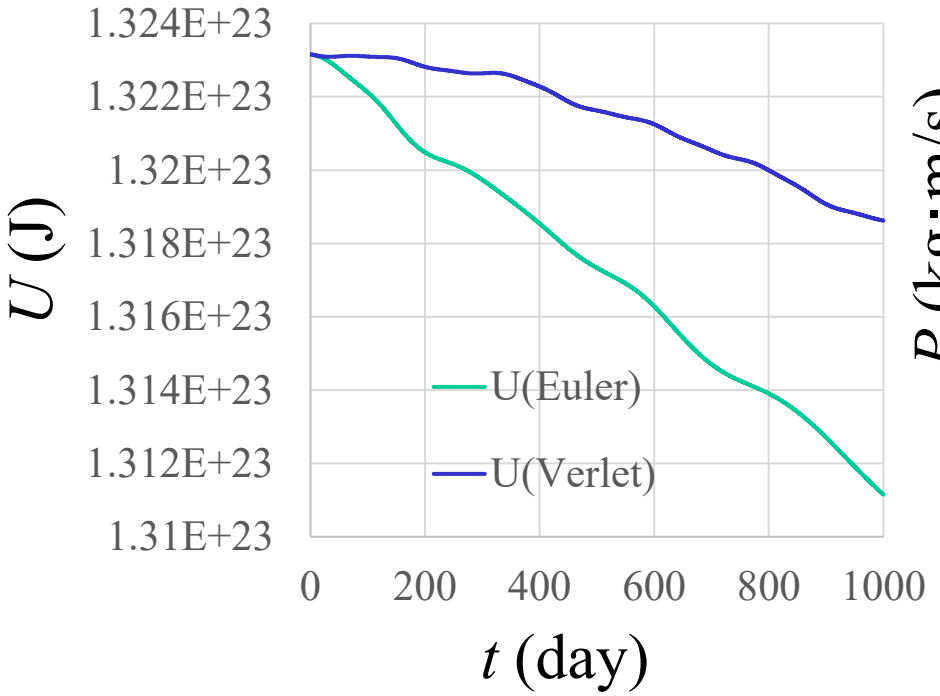


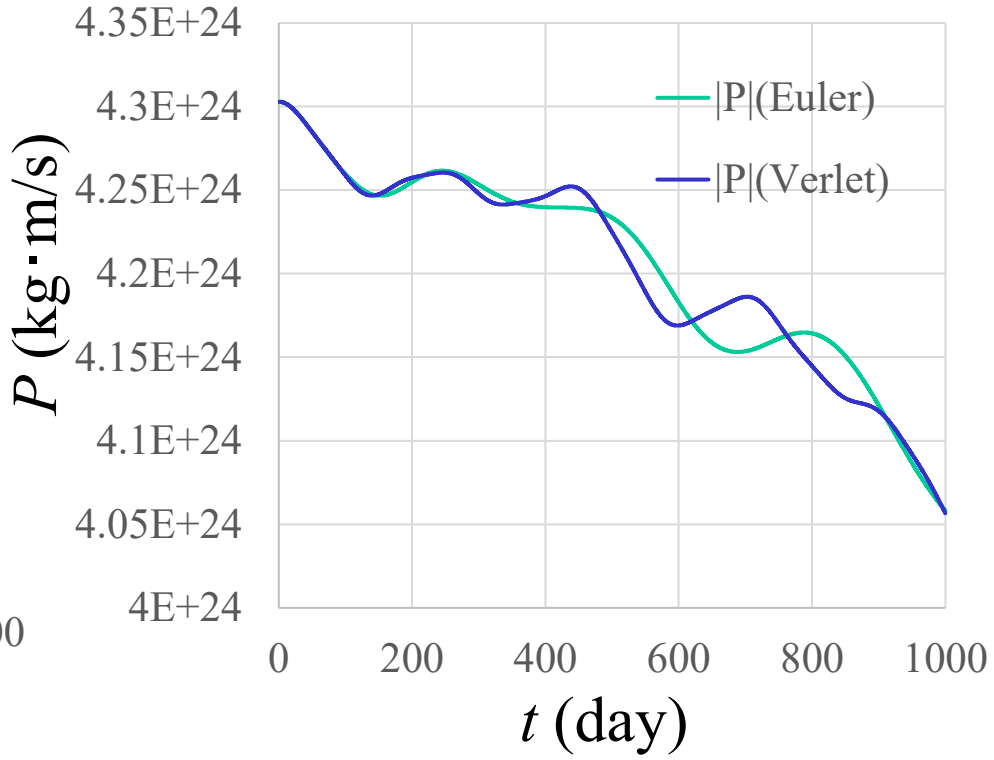
python diffeq2nd_planet.py **Verlet** 0.2 5000



Program: Check by conservation law

> python diffeq2nd_planet.py Euler 0.2 5000
> python diffeq2nd_planet.py Verlet 0.2 5000





Speed up python: Planet simulation

1. List : diffeq2nd_planet.py
2. numpy.ndarray : diffeq2nd_planet_generator.py
3. numba (parallel=True) : diffeq2nd_planet_generator_numba.py
4. numba (parallel=False) : diffeq2nd_planet_generator_numba.py
5. C DLL : diffeq2nd_planet_c.py

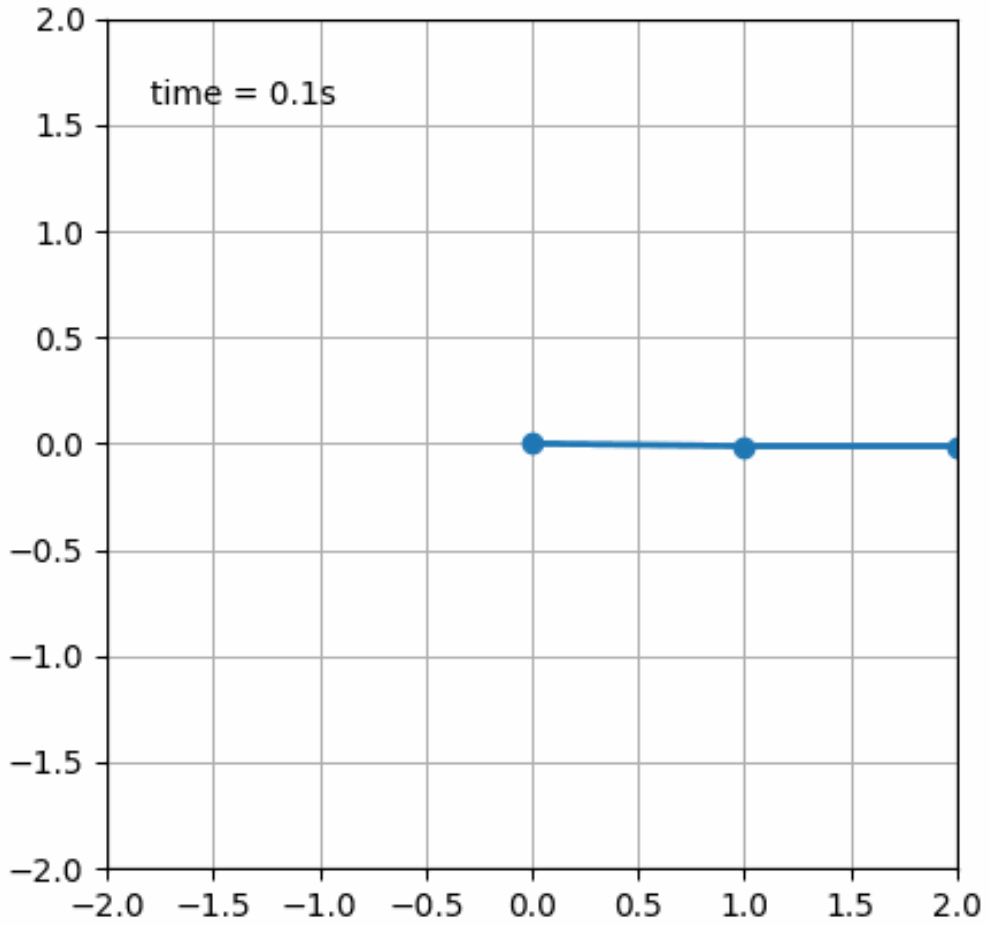
Implementation	Time [s]	First 1-step run for numba JIT compilation
List	220.594	
Numpy	37.553	
Numba (parallel=True, 4 core/8 threads)	41.305	3.909
Namba (parallel=False)	44.170	0.978
C DLL	48.262	

- Use numpy instead of python list variables and explicit python loops.
- Numpy may outperform a simple C DLL implementation because its internal routines are highly optimized.
- Numba just-in-time (JIT) compilation is another option

Program: Double pendulum (2重振り子)

> python double_pendulum_ode.py

```
from scipy.integrate import odeint  # Ordinaly Differential Equation Integrator

def derivs(state, t):
    dtheta1_dt = state[2]
    dtheta2_dt = state[3]

    delta = state[1] - state[0]

    denominator1 = (m1 + m2) * L1 - m2 * L1 * np.cos(delta) * np.cos(delta)
    denominator2 = (L2 / L1) * denominator1

    dtheta1_dot_dt = ((m2 * L1 * state[2] ** 2 * np.sin(delta) * np.cos(delta) +
                       m2 * g * np.sin(state[1]) * np.cos(delta) +
                       m2 * L2 * state[3] ** 2 * np.sin(delta) -
                       (m1 + m2) * g * np.sin(state[0])) / denominator1)

    dtheta2_dot_dt = ((-m2 * L2 * state[3] ** 2 * np.sin(delta) * np.cos(delta) +
                       (m1 + m2) * g * np.sin(state[0]) * np.cos(delta) -
                       (m1 + m2) * L1 * state[2] ** 2 * np.sin(delta) -
                       (m1 + m2) * g * np.sin(state[1])) / denominator2)

    return [dtheta1_dt, dtheta2_dt, dtheta1_dot_dt, dtheta2_dot_dt]

state = [theta1, theta2, theta1_dot, theta2_dot]
t = np.arange(0.0, tmax, tstep)

y = odeint(derivs, state, t)
```



Packet collision simulator

Harmonic: > python md_packets_FuncAnimation.py --initial_mode two_packets --scatterer_mode none

Anharmonic: > python md_packets_FuncAnimation.py --initial_mode two_packets --scatterer_mode none --k4 500