Numerical Methods for Geodynamo Simulation

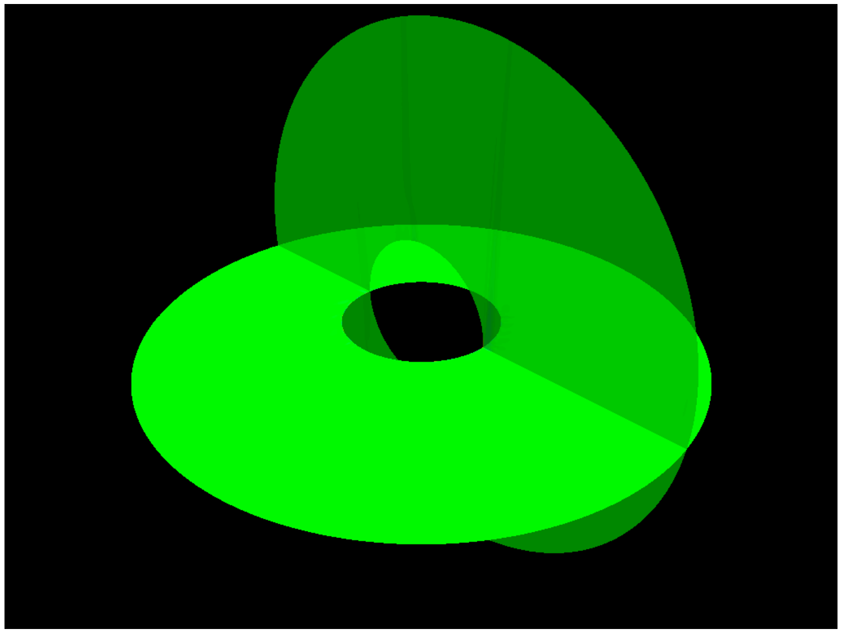
Akira Kageyama
Earth Simulator Center, JAMSTEC, Japan

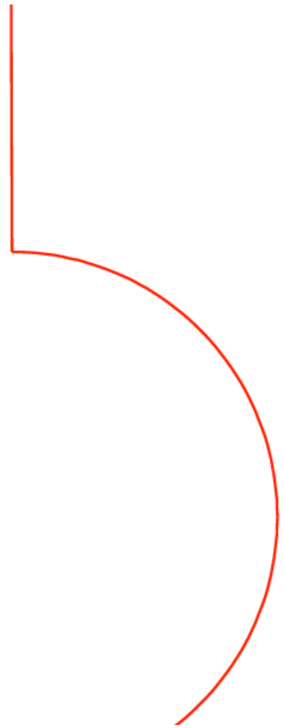
Part 1

2007.08 Les Houches Summer School "Dynamos"

Goal of these lectures

- Basics of numerical methods
  - Finite difference method + Runge-Kutta integration
- To make your own computer simulation code by FDM+RK





Outline

- Speed of computers
- Finite difference method
- Numerical integration
- Simple sample 1-D simulation

Sorce codes:

source_codes.tar.gz

Floating point number

- Double precision (64 bit)
  - 1 bit for sign.
  - 11 bits for exponent.
  - 52 bits for mantissa.

$$2^{52} \sim 10^{15.6}$$

- π in double precision floating point

$$\pi = 3.14159265358979$$

- FLOPS = Floating Point Operations Per Second
- Operatios
  - multiplication (division)
  - addition (subtraction)

What is Your FLOPS value?

- How many seconds do you need to calculate using only a pen and paper?

$$
\begin{array}{r}
\pi^2 = 3.14159265358979 \\
\times 3.14159265358979 \\
\hline
?.??????????????
\end{array}
$$

- … 1000 seconds?
- ……then, you are a 0.001 FLOPS computer.

Human brain as a computer

If a person

- has 0.001 FLOPS ability,
- lives for 100 years,
- devotes the entire life (without sleep) to the floating point number operations (multiplication/addition),

then only 3 million operations are the life work.

$$\left(100 \text{ years} = 3 \times 10^{9} \text{ seconds}\right) \times 0.001$$

How about this PC?

```
program countflops
  implicit none
  integer, parameter :: SP = kind(1.0)
  integer, parameter :: DP = selected_real_kind(2*precision(1.0_SP))
  real(DP), parameter :: PI = 3.14159265358979_DP
  real(DP) :: a
  integer :: i

  do i = 1 , 3*(10**6)
     a = PI*PI
  end do

  print *, ' a = ', a

end program countflops
```

In sourcecodes_tar.gz,
- src/CountFlops/

Result

- 3 million floating point operations
- A human takes 100 years.
- … 0.07 seconds by this PC.
- $\frac{3\times10^6}{0.07} = 4 \times 10^7$
- ===> 40 M FLOPS (with this iBook).
- Could be 1 G FLOPS

Top500 Supercomputer List

http://www.top500.org/

Home › Lists › June 2007

TOP500 List - June 2007 (1-100)

Rank	Computer	Processors	R_{peak}
1	BlueGene/L - eServer Blue Gene Solution IBM	131072	367000
2	Jaguar - Cray XT4/XT3 Cray Inc.	23016	119350
3	Red Storm - Sandia/ Cray Red Storm, Opteron 2.4 GHz dual core Cray Inc.	26544	127411
4	BGW - eServer Blue Gene Solution IBM	40960	114688
5	New York Blue - eServer Blue Gene Solution IBM	36864	103219
6	ASC Purple - eServer pSeries p5 575 1.9 GHz IBM	12208	92781

367 TFLOPS

$$1\mathrm{T} = 10^{12}$$

$$100\,\mathrm{T} = 10^{14}$$

Speed contrast

- Today's PC = GFLOPS (10^9 FLOPS)
- Today's supercomputer = 10^{14} FLOPS
- Supercomputer is 10^5 times faster than PC.

Speed contrast

- Today's PC = GFLOPS (10^9 FLOPS)
- Today's supercomputer = 10^{14} FLOPS
- Supercomputer is 10^5 times faster than PC.
- Concorde is only 400-500 times faster than our walk speed.

How it could be done?

- Each processor does not have super-FLOPS.
- Parallelization.
  - A set of many processors makes a whole computer.

Top500 Supercomputer List

http://www.top500.org/

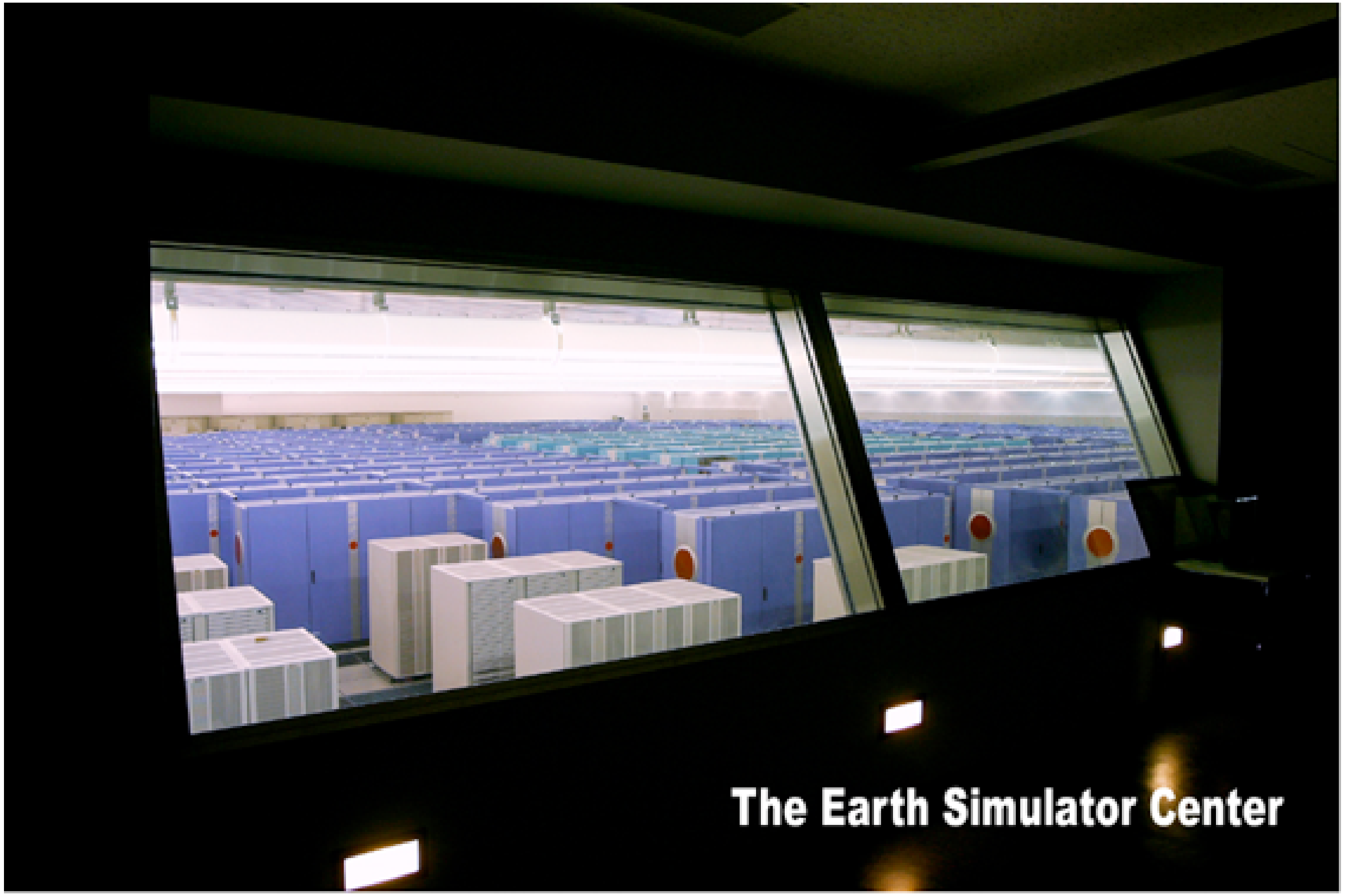
Home › Lists › June 2007

TOP500 List - June 2007 (1-100)

Rank	Computer	Processors	R_{peak}
1	BlueGene/L - eServer Blue Gene Solution IBM	131072	367000
2	Jaguar - Cray XT4/XT3 Cray Inc.	23016	119350
3	Red Storm - Sandia/ Cray Red Storm, Opteron 2.4 GHz dual core Cray Inc.	26544	127411
4	BGW - eServer Blue Gene Solution IBM	40960	114688
5	New York Blue - eServer Blue Gene Solution IBM	36864	103219
6	ASC Purple - eServer pSeries p5 575 1.9 GHz IBM	12208	92781

> 130k processors

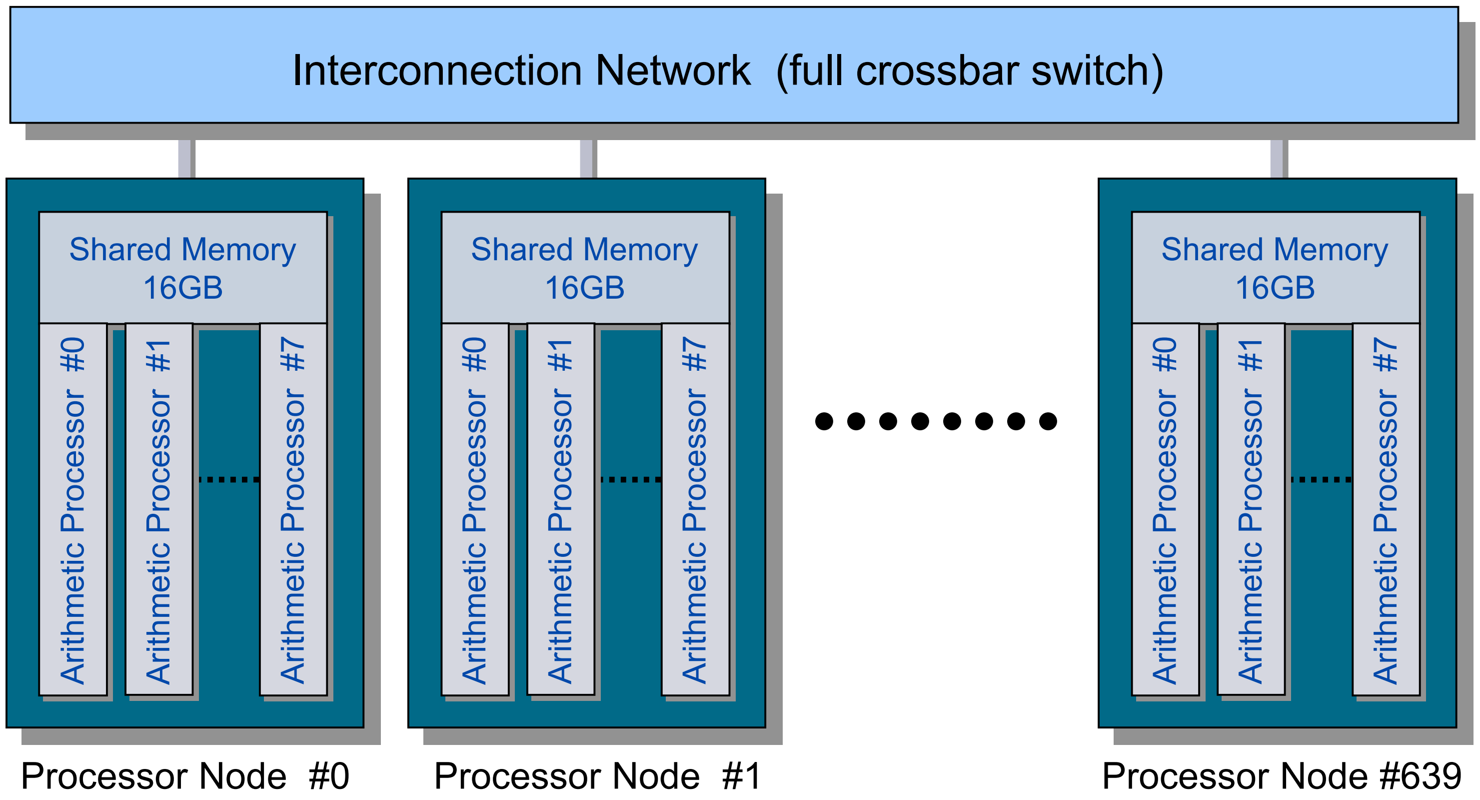
Earth Simulator in Japan



Earth Simulator in Japan

- Peak performance/AP : 8Gflops
- Peak performance/PN : 64Gflops
- Shared memory/PN : 16GB

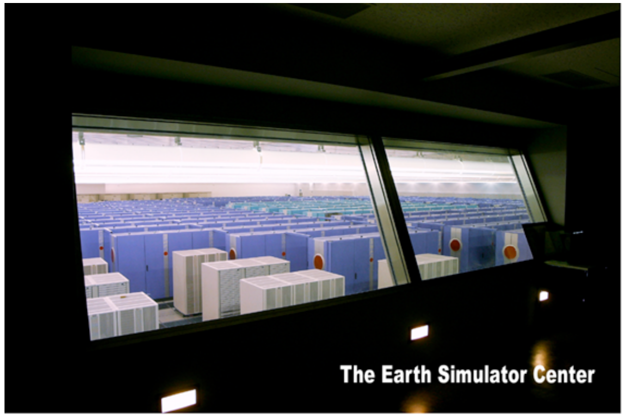
- Total number of APs : 5120
- Total number of PNs : 640
- Total peak performance : 40TFLOPS
- Total main memory : 10TB



Challenge of massively parallel computing

τ_1 : simulaiton time by 1 processor

τ_P : simulaiton time by P processors

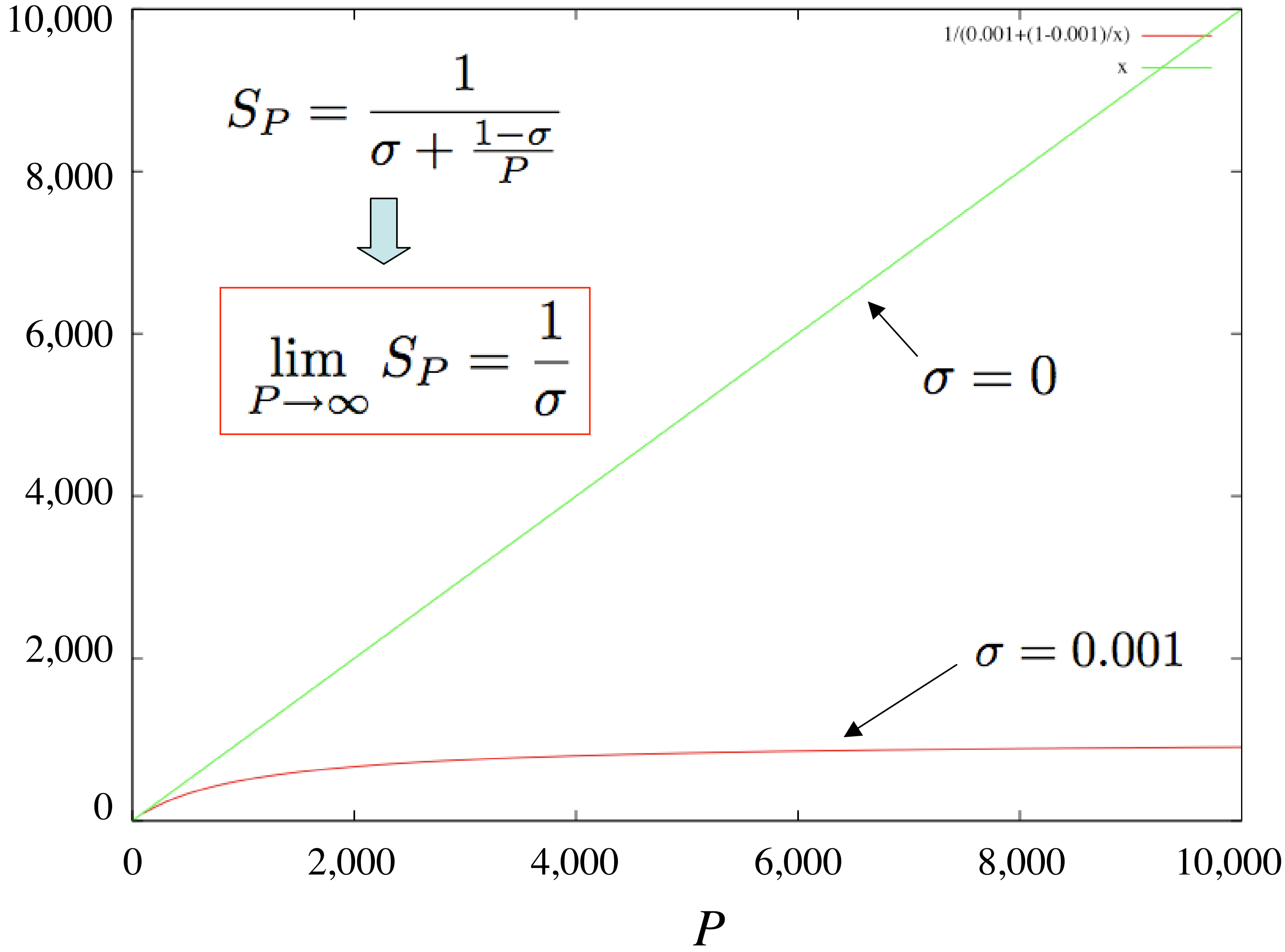


$\sigma\,\tau_1$:	Parallelization impossible
$(1-\sigma)\,\tau_1$:	Parallelization possible

Acceleration by parallel processing by *P* processors:

$$S_P = \frac{1}{\sigma + \frac{1-\sigma}{P}}$$

Challenge of massively parallel computing

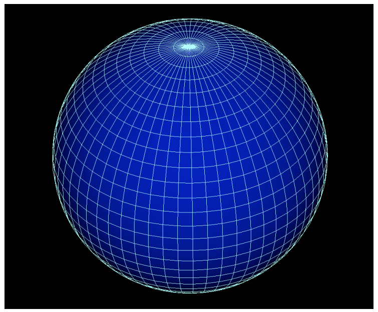


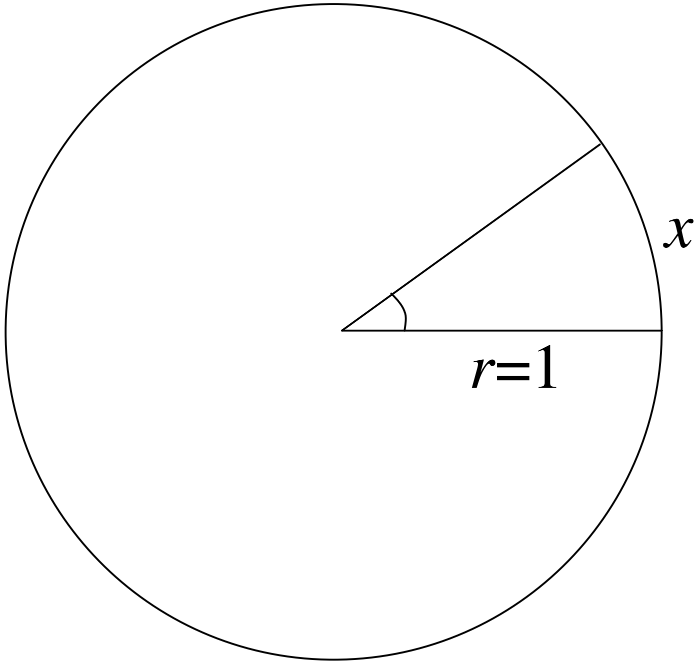
Methods used in geodynamo simulations

- Spectral-based methods
  - Spherical harmonics expansion
  - Double Fourier expansion
  - Beltrami function expansion
- Other methods
  - Finite difference method
  - Finite volume method
  - Finite element method
  - Cartesian grid method

Finite Difference Method (FDM) :
An explanation through a simple 1-D problem

The diffusion equation on a circle



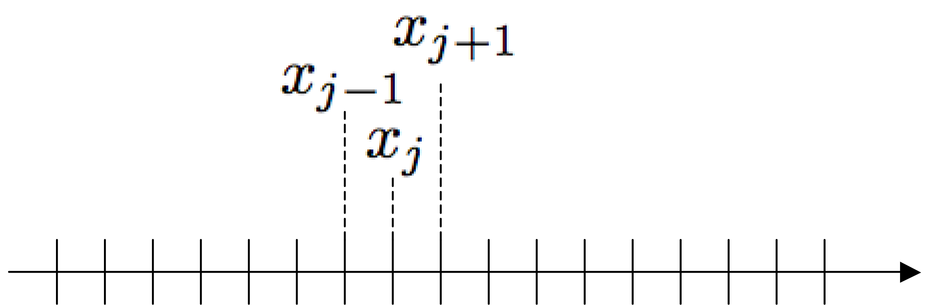


$$\frac{\partial \psi}{\partial t} = \frac{\partial^2 \psi}{\partial x^2}$$

$$0 \leq x < 2\pi$$

$$\frac{\partial \psi}{\partial t} = \frac{\partial^2 \psi}{\partial x^2}$$

FDM: Finite Difference Method



$$\frac{d\psi}{dx} = \frac{\psi_{j+1} - \psi_{j-1}}{2\Delta x} + O(\Delta x)^2$$

$$\frac{d^2\psi}{dx^2} = \frac{\psi_{j+1} - 2\psi_j + \psi_{j-1}}{(\Delta x)^2} + O(\Delta x)^2$$

$$\frac{d\psi_j}{dt} = \frac{\psi_{j+1} - 2\psi_j + \psi_{j-1}}{(\Delta x)^2}$$

FDM: Finite Difference Method

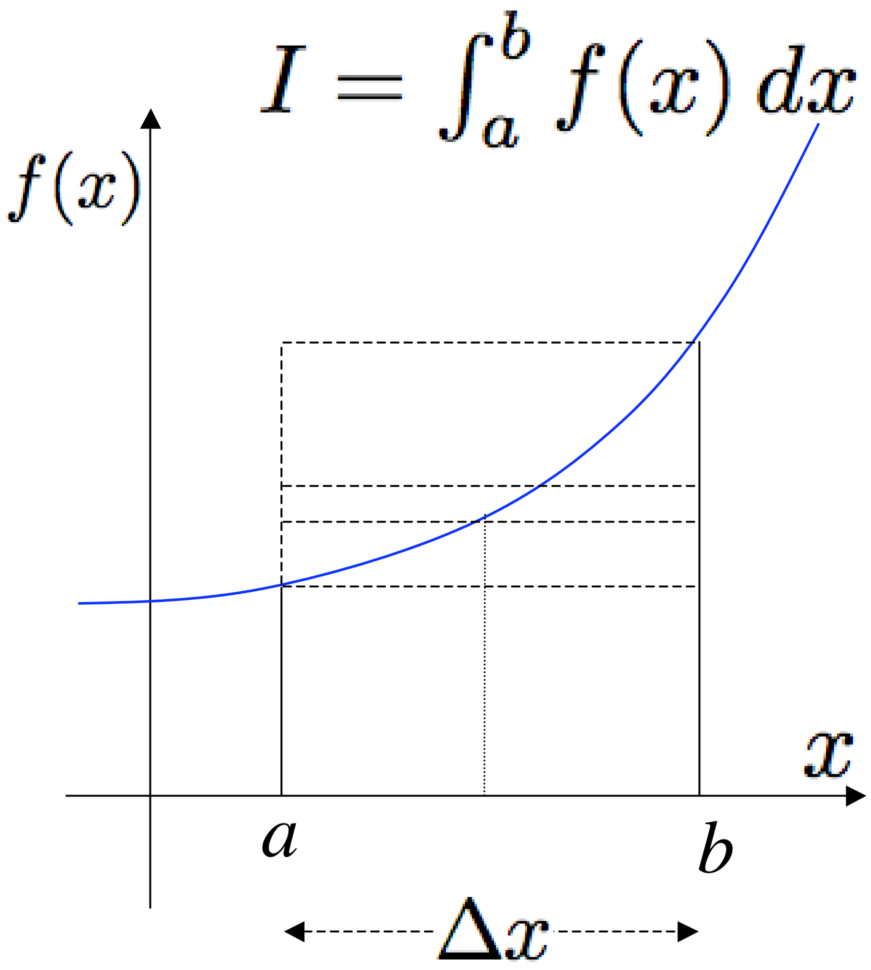
$$\frac{\partial \psi}{\partial t} = \frac{\partial^2 \psi}{\partial x^2}$$

$$\frac{d\psi_j}{dt} = \frac{\psi_{j+1} - 2\psi_j + \psi_{j-1}}{(\Delta x)^2}$$

$$\frac{d\psi_j}{dt} = f(\psi_1, \psi_2, \cdots, \psi_N)$$

==> Time integration.

Numerical integration



1) $I = \Delta x\, f(a)$

Error $\propto O(\Delta x^2)$

2) Trapezoid rule

$$I = \frac{\Delta x}{2}\,[f(a) + f(b)]$$

Error $\propto O(\Delta x^3)$

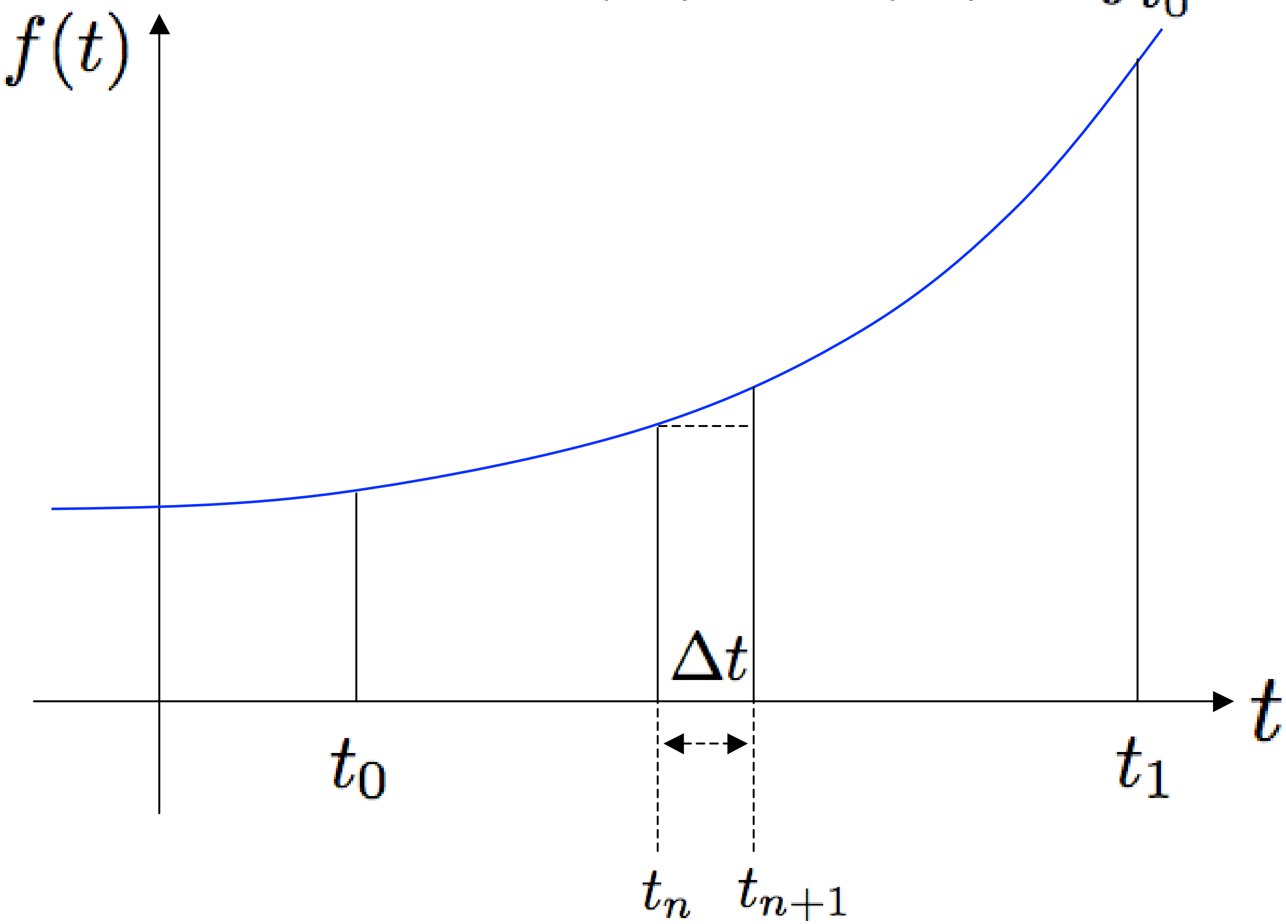
3) Simpson's rule

$$I = \frac{\Delta x}{6}\,\left[f(a) + 4f\left(\tfrac{a+b}{2}\right) + f(b)\right]$$

Error $\propto O(\Delta x^5)$

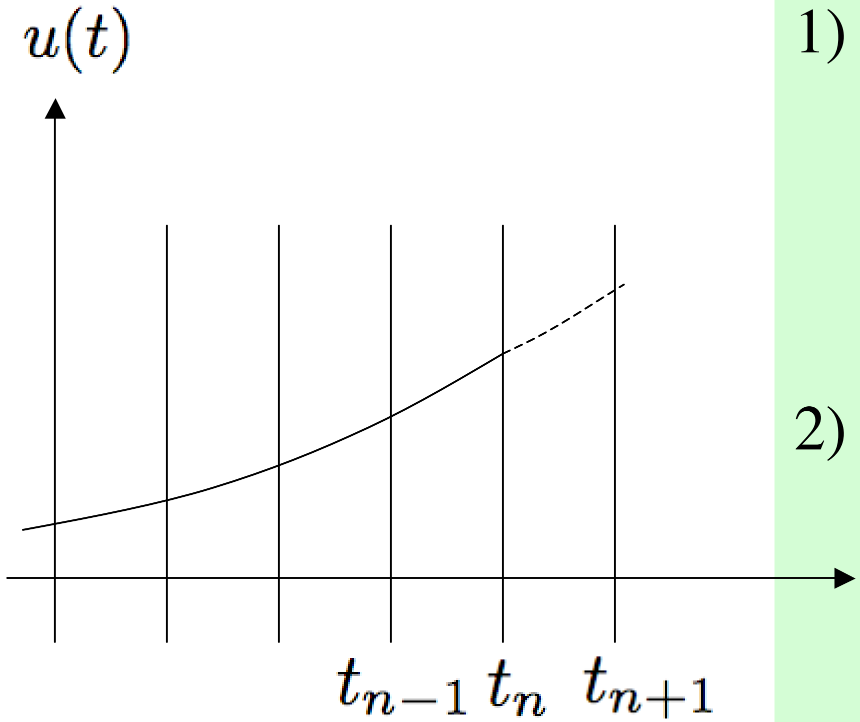
$$\frac{du(t)}{dt} = f(u(t), t)$$

$$u(t_1) = u(t_0) + \int_{t_0}^{t_1} f\, dt$$



$$\frac{du(t)}{dt} = f(u(t), t)$$

$$u(t_{n+1}) = u(t_n) + \int_{t_n}^{t_{n+1}} f\, dt$$



1)

$$u(t_{n+1}) = u(t_n) + \Delta t\, f(t_n)$$

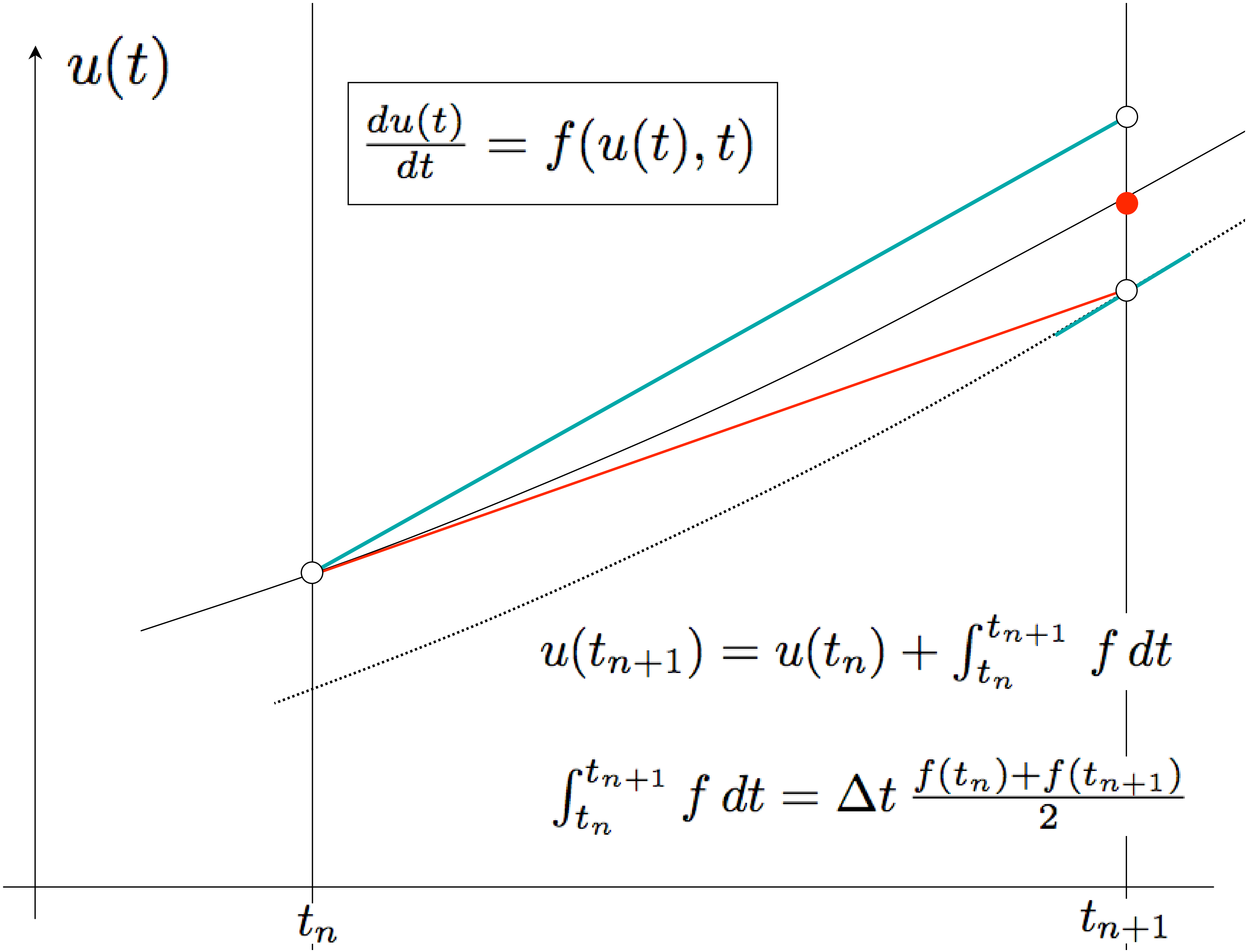
⇨ 1st order Euler method

2) Trapezoid rule

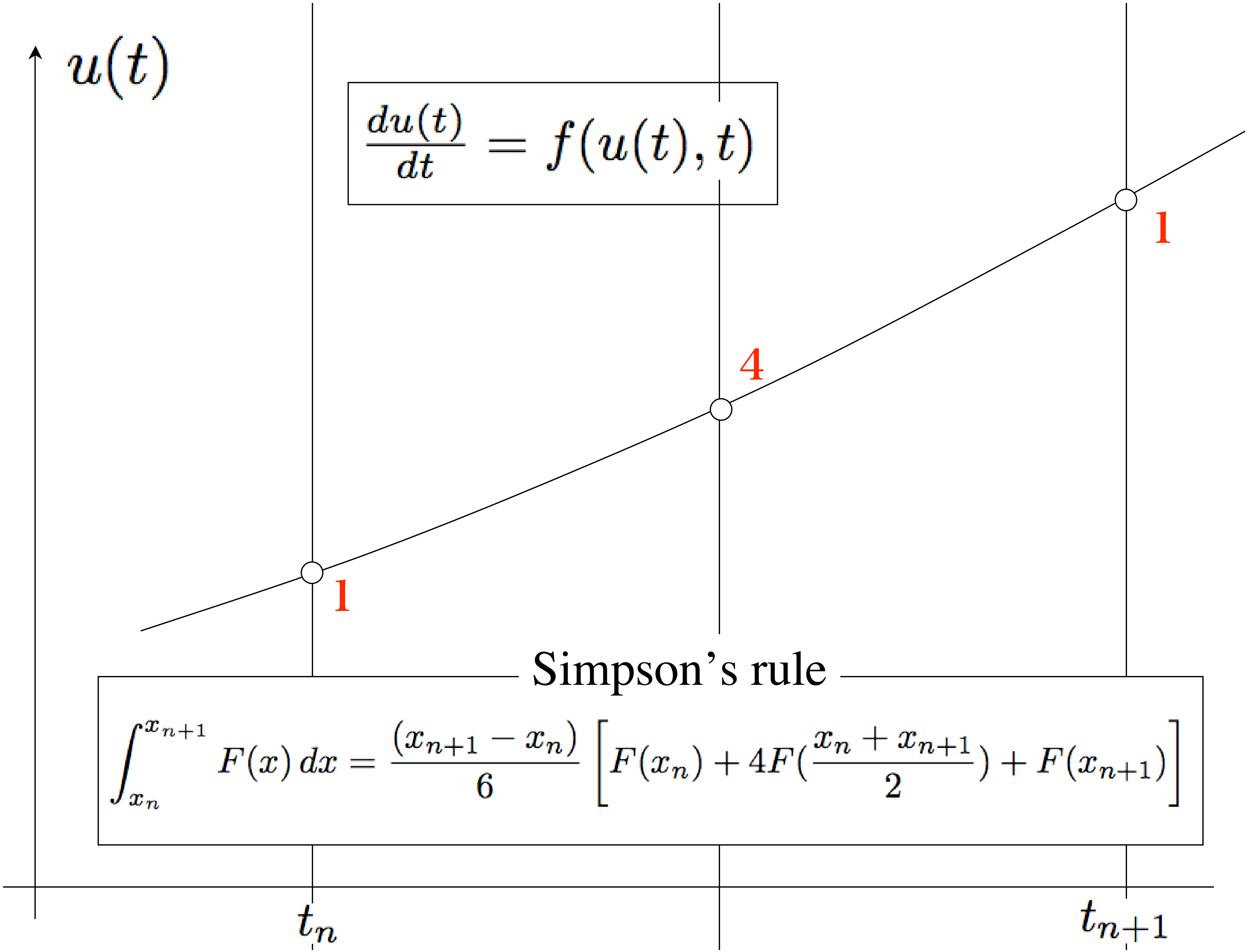
⇨ 2nd order Runge-Kutta method

3) Simpson's rule

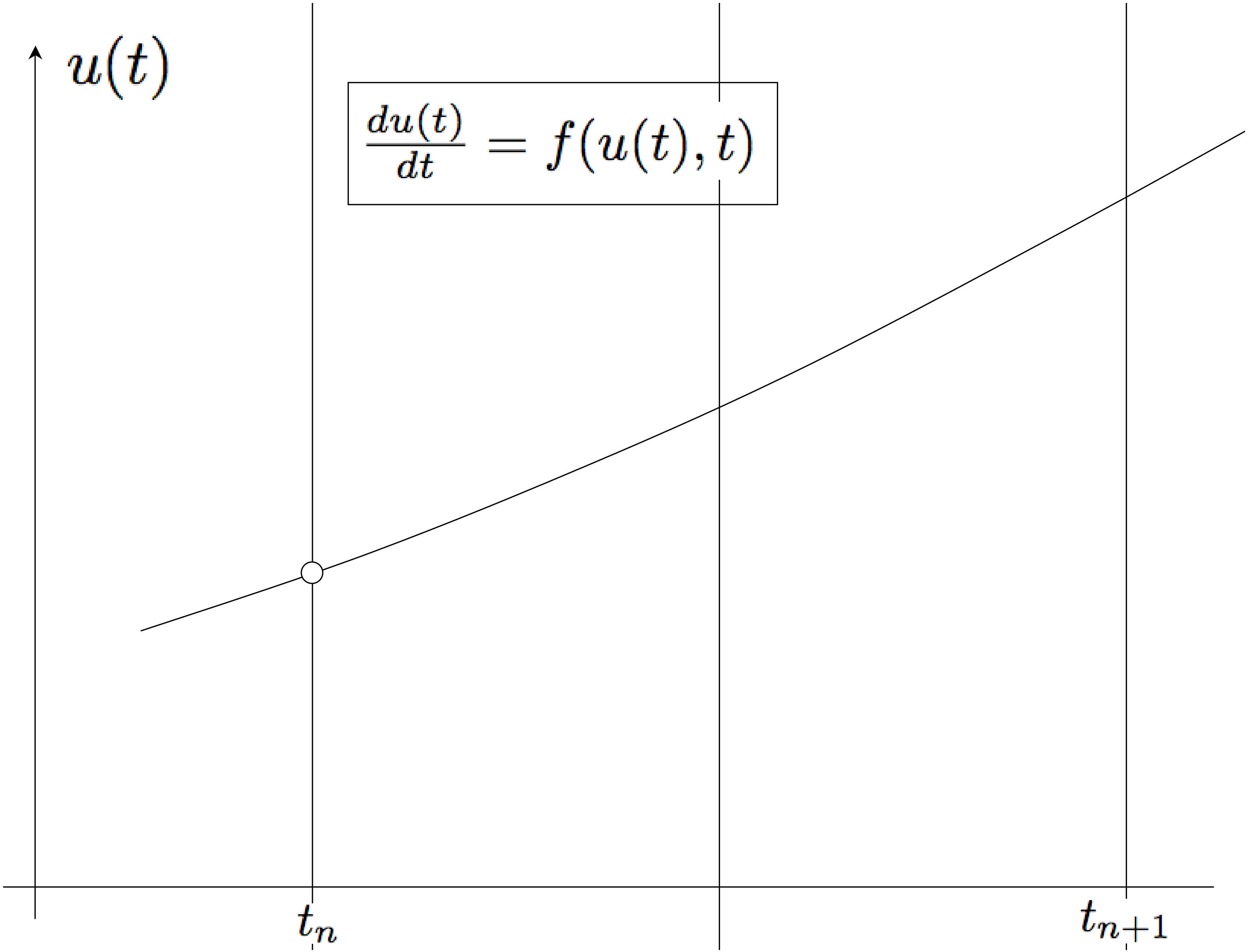
⇨ 4th order Runge-Kutta method

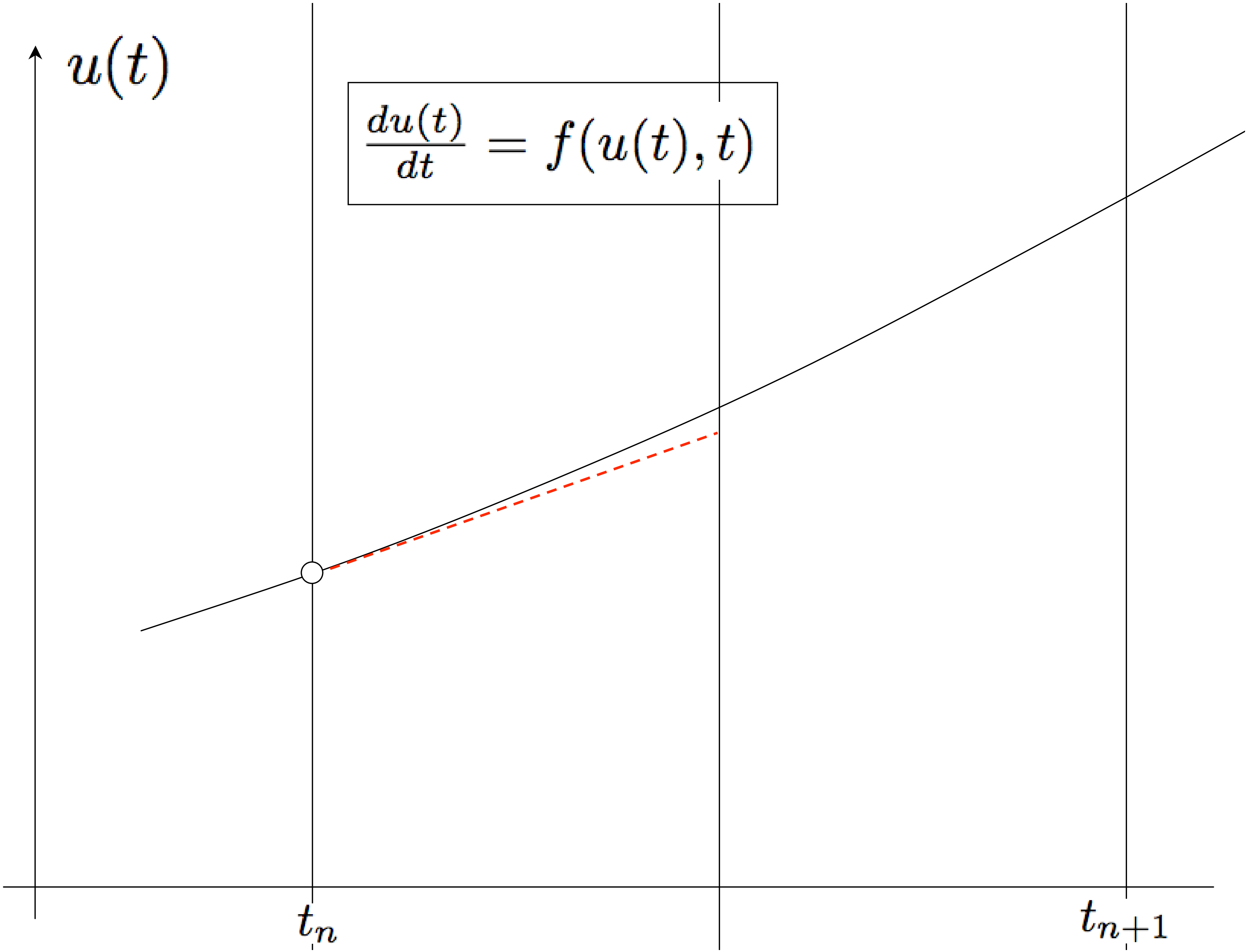


$u(t)$
$\frac{du(t)}{dt} = f(u(t), t)$
$u(t_{n+1}) = u(t_n) + \int_{t_n}^{t_{n+1}} f\,dt$
$\int_{t_n}^{t_{n+1}} f\,dt = \Delta t\,\frac{f(t_n)+f(t_{n+1})}{2}$
t_n
t_{n+1}

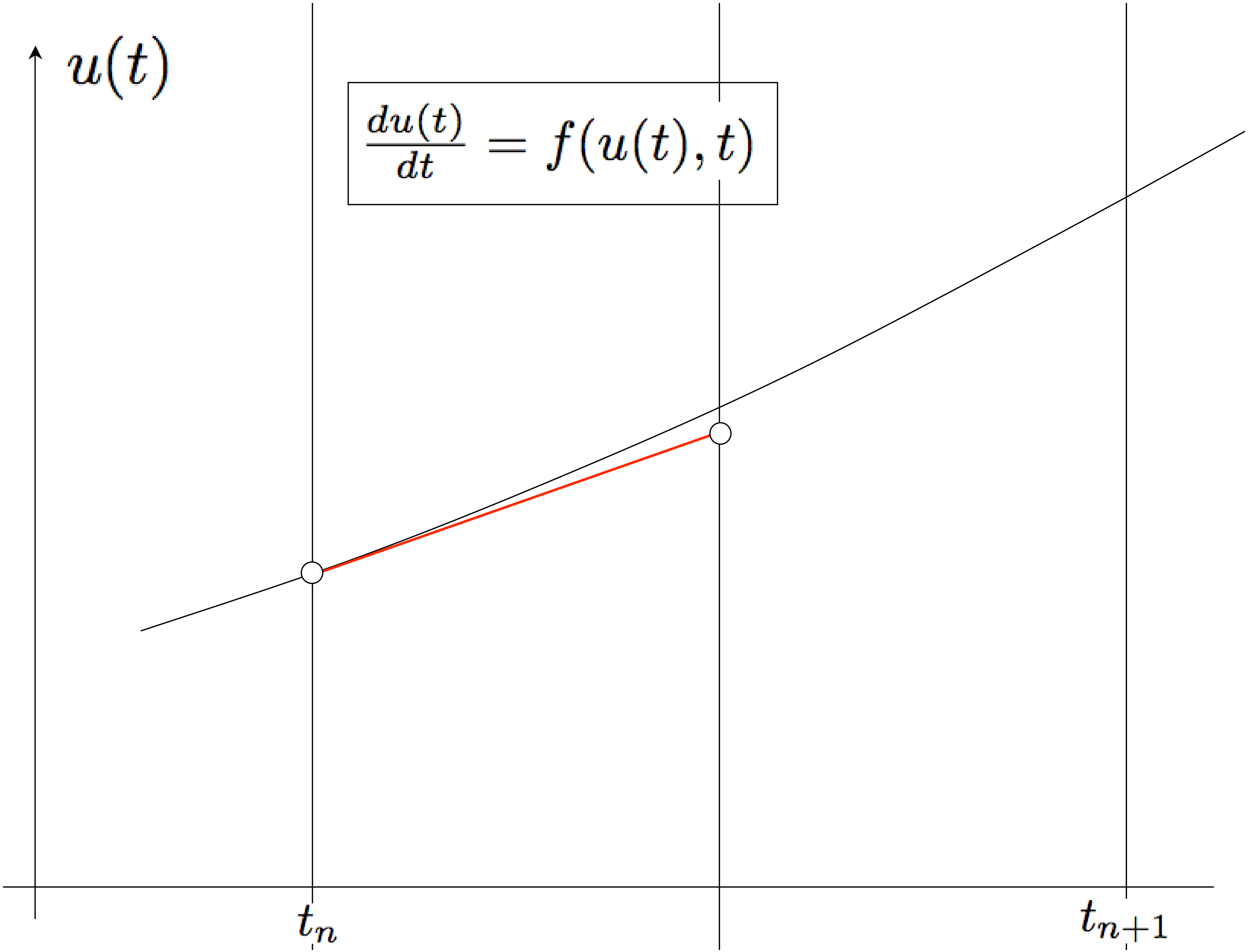


$u(t)$
$\frac{du(t)}{dt} = f(u(t), t)$
1
4
1
Simpson's rule
$\int_{x_n}^{x_{n+1}} F(x)\,dx = \frac{(x_{n+1} - x_n)}{6}\left[F(x_n) + 4F(\frac{x_n + x_{n+1}}{2}) + F(x_{n+1})\right]$
t_n
t_{n+1}

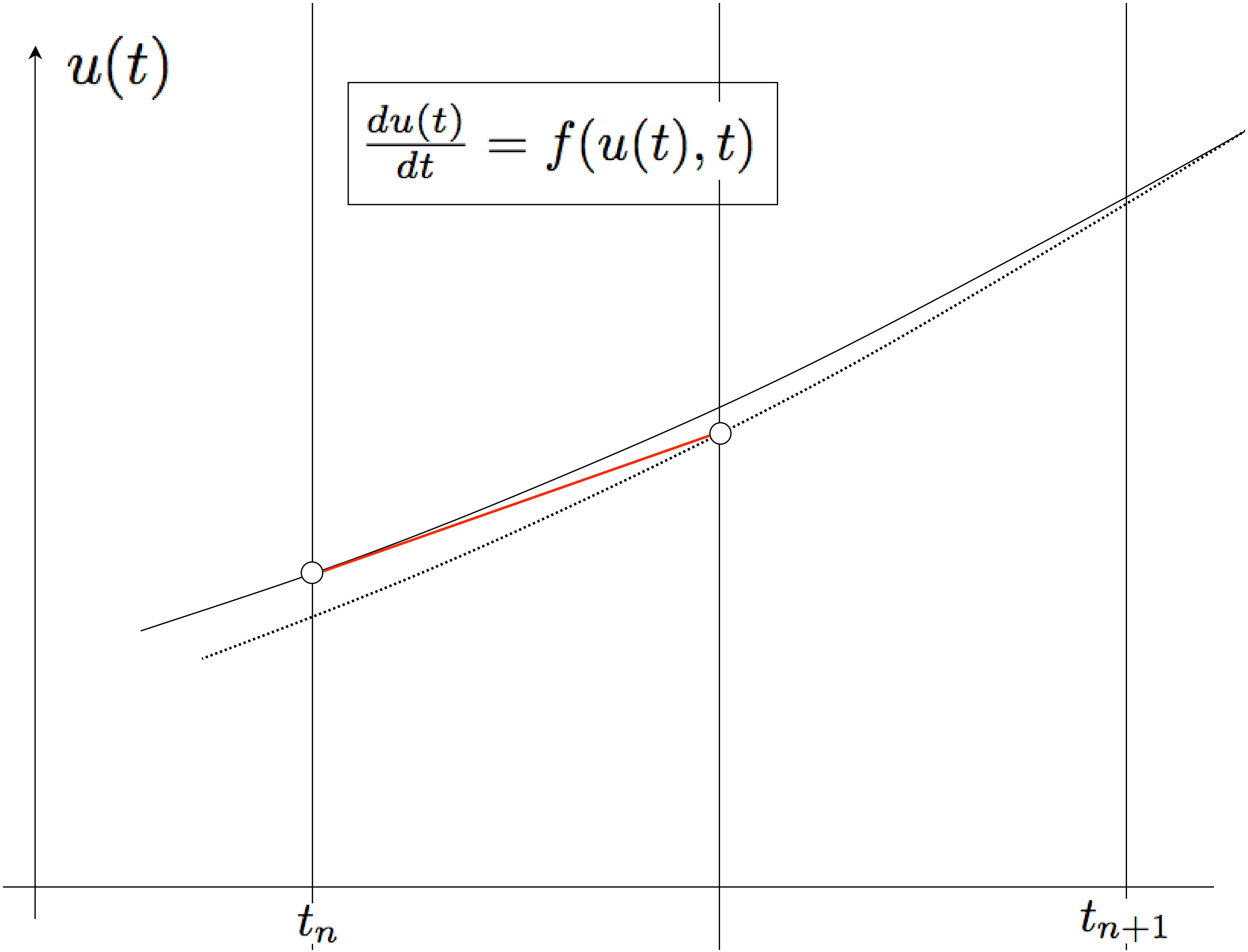

$u(t)$
$\frac{du(t)}{dt} = f(u(t), t)$
t_n
t_{n+1}



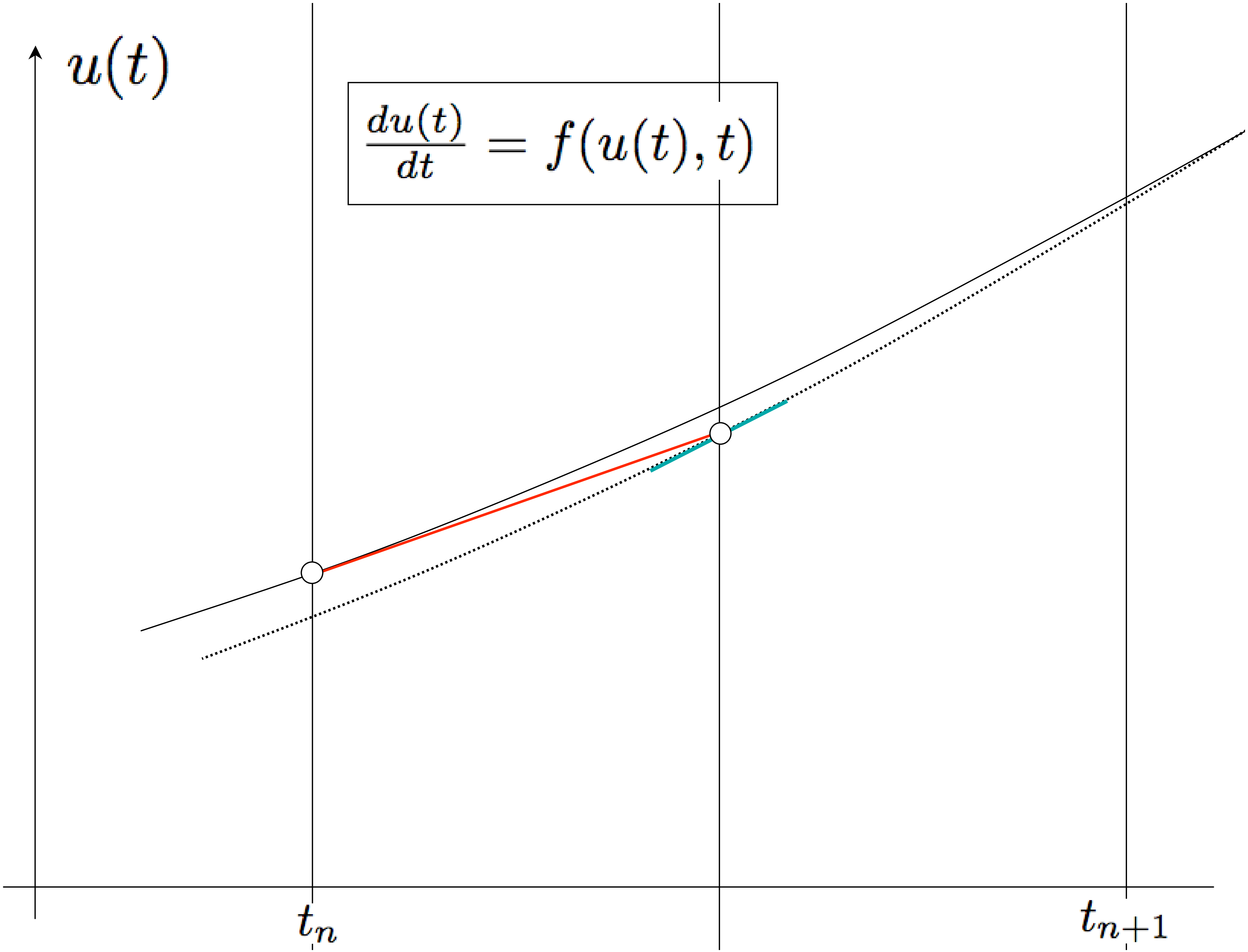
u(t)
du(t)/dt = f(u(t), t)
t_n
t_{n+1}

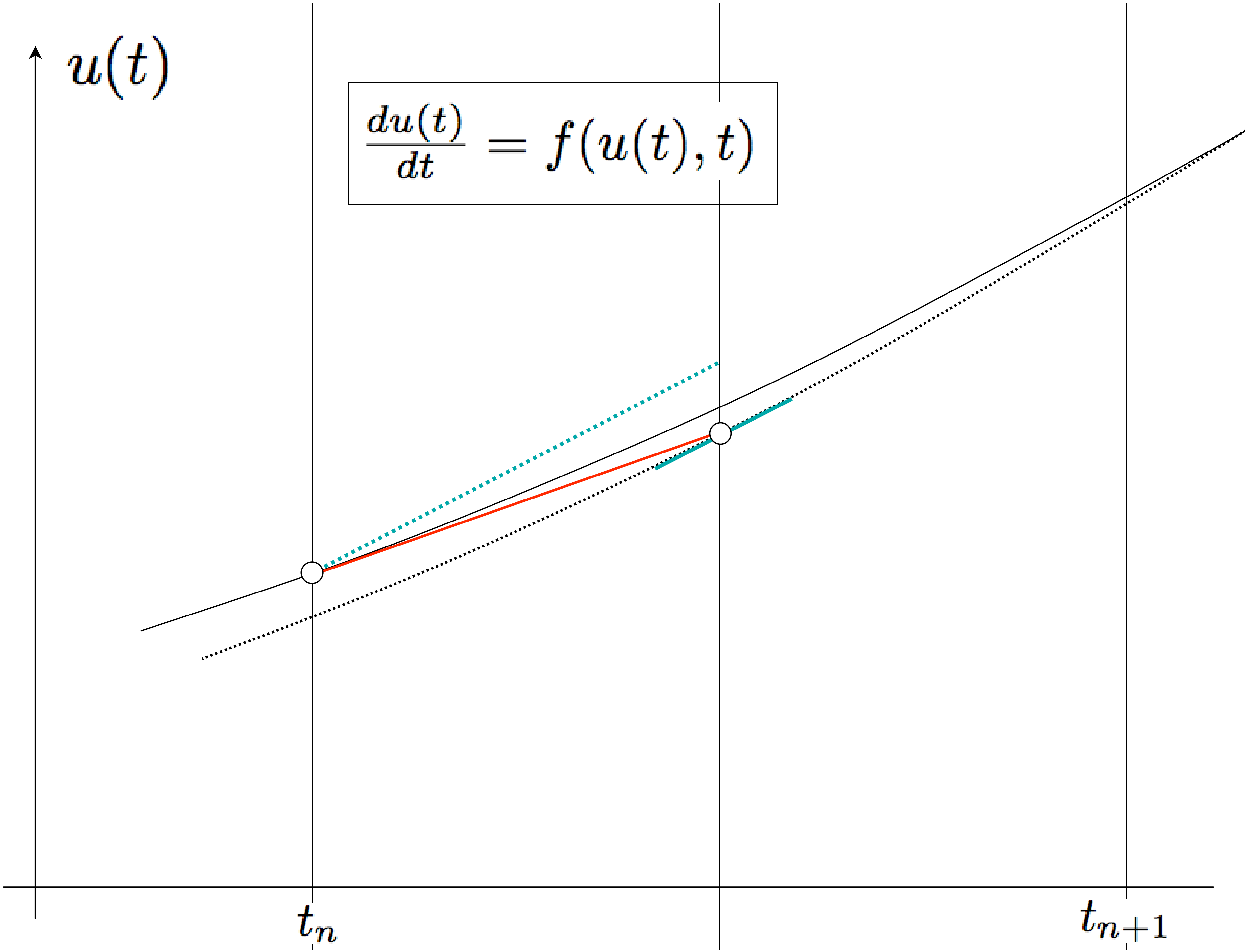


u(t)
du(t)/dt = f(u(t), t)
t_n
t_{n+1}

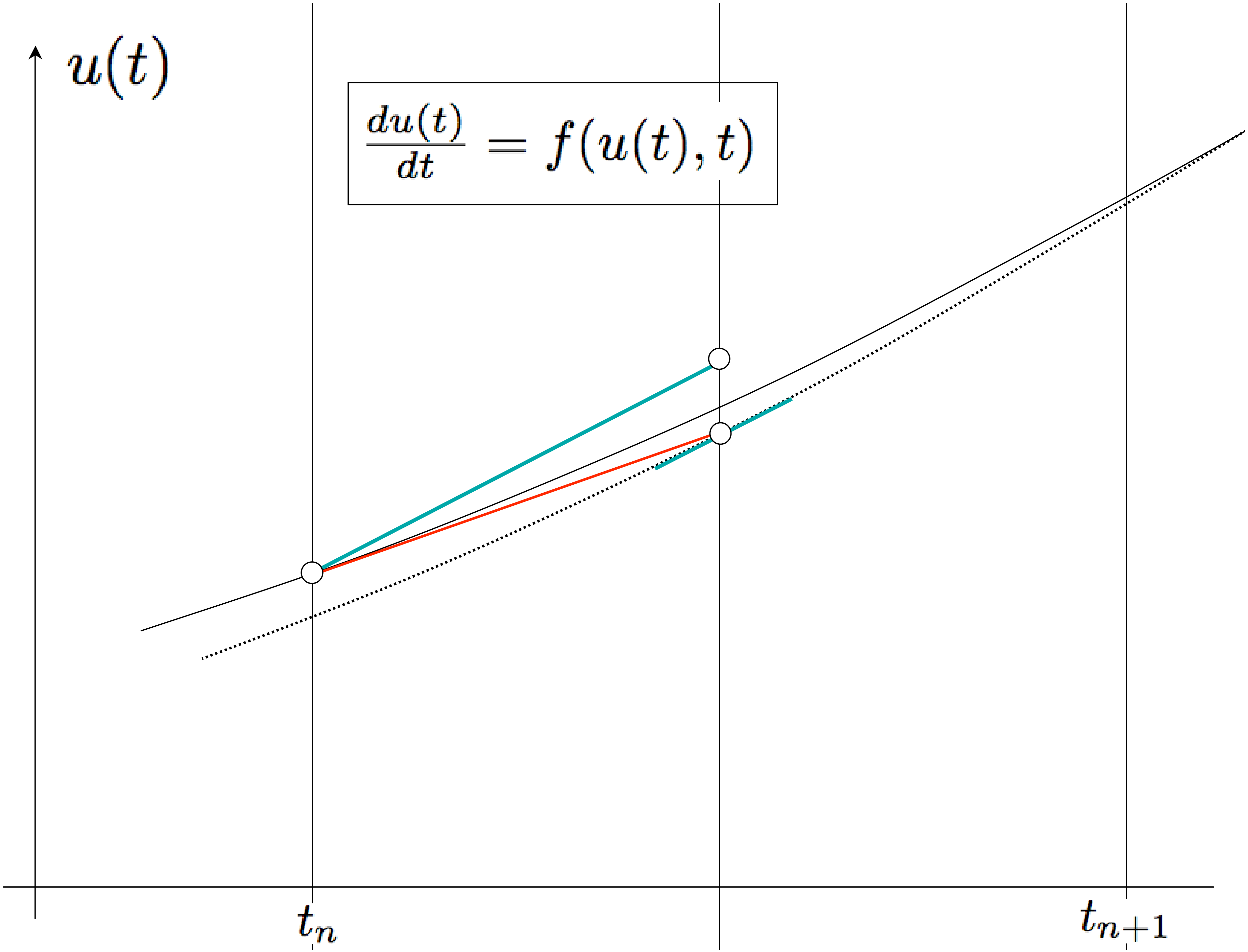


u(t)
du(t)/dt = f(u(t), t)
t_n
t_{n+1}

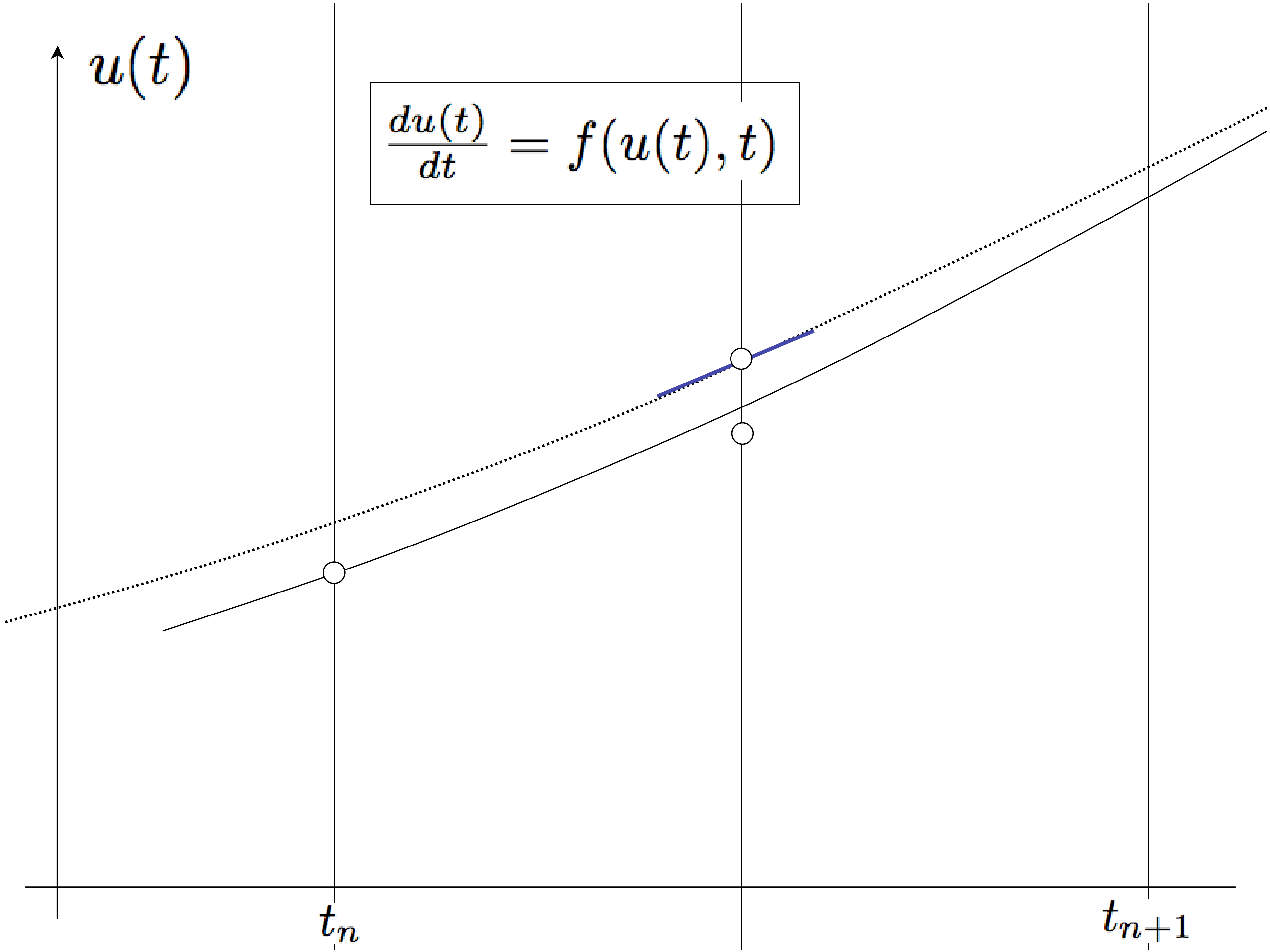

$u(t)$
$\frac{du(t)}{dt} = f(u(t), t)$
t_n
t_{n+1}



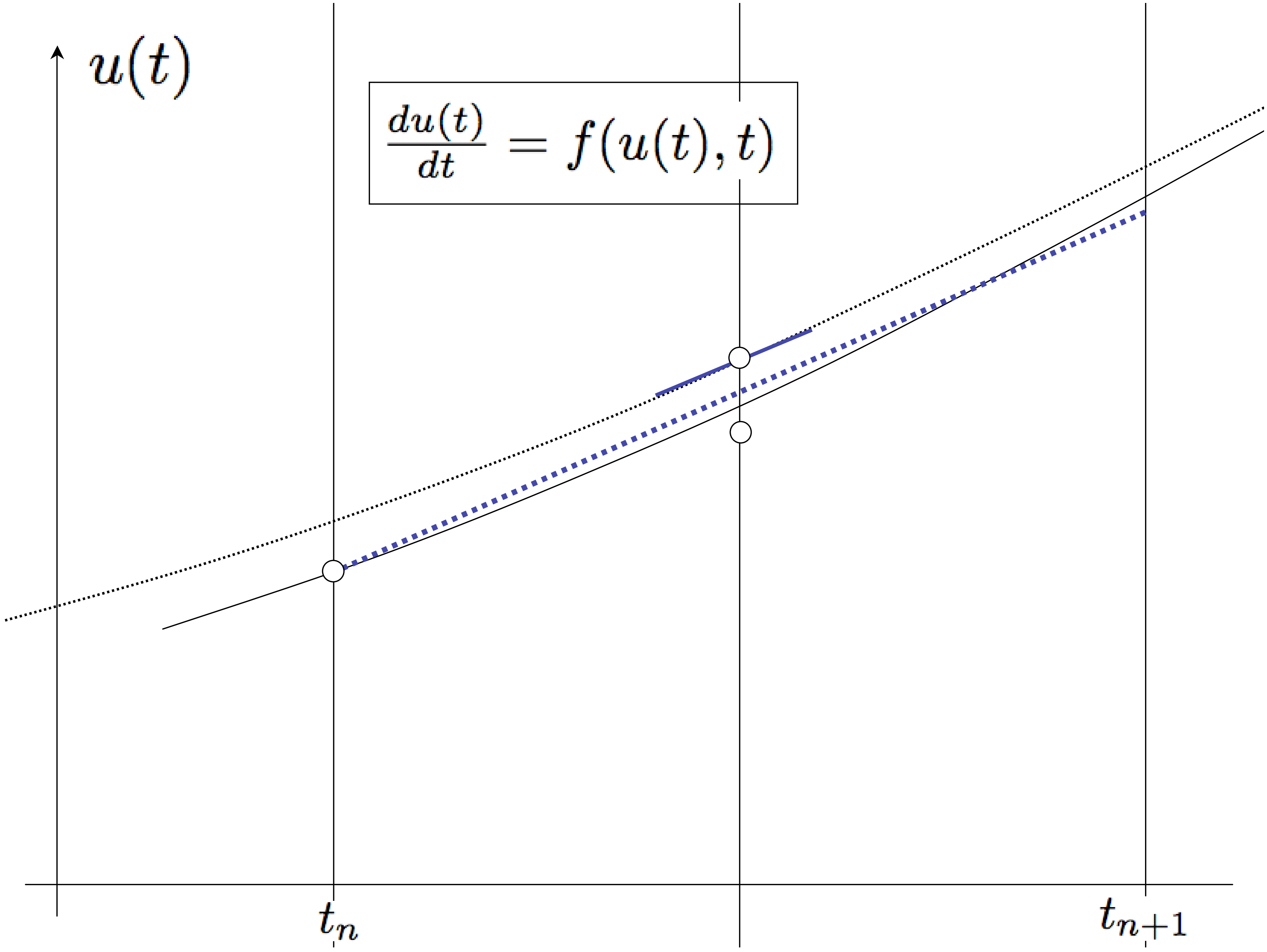
$u(t)$
$\frac{du(t)}{dt} = f(u(t), t)$
t_n
t_{n+1}



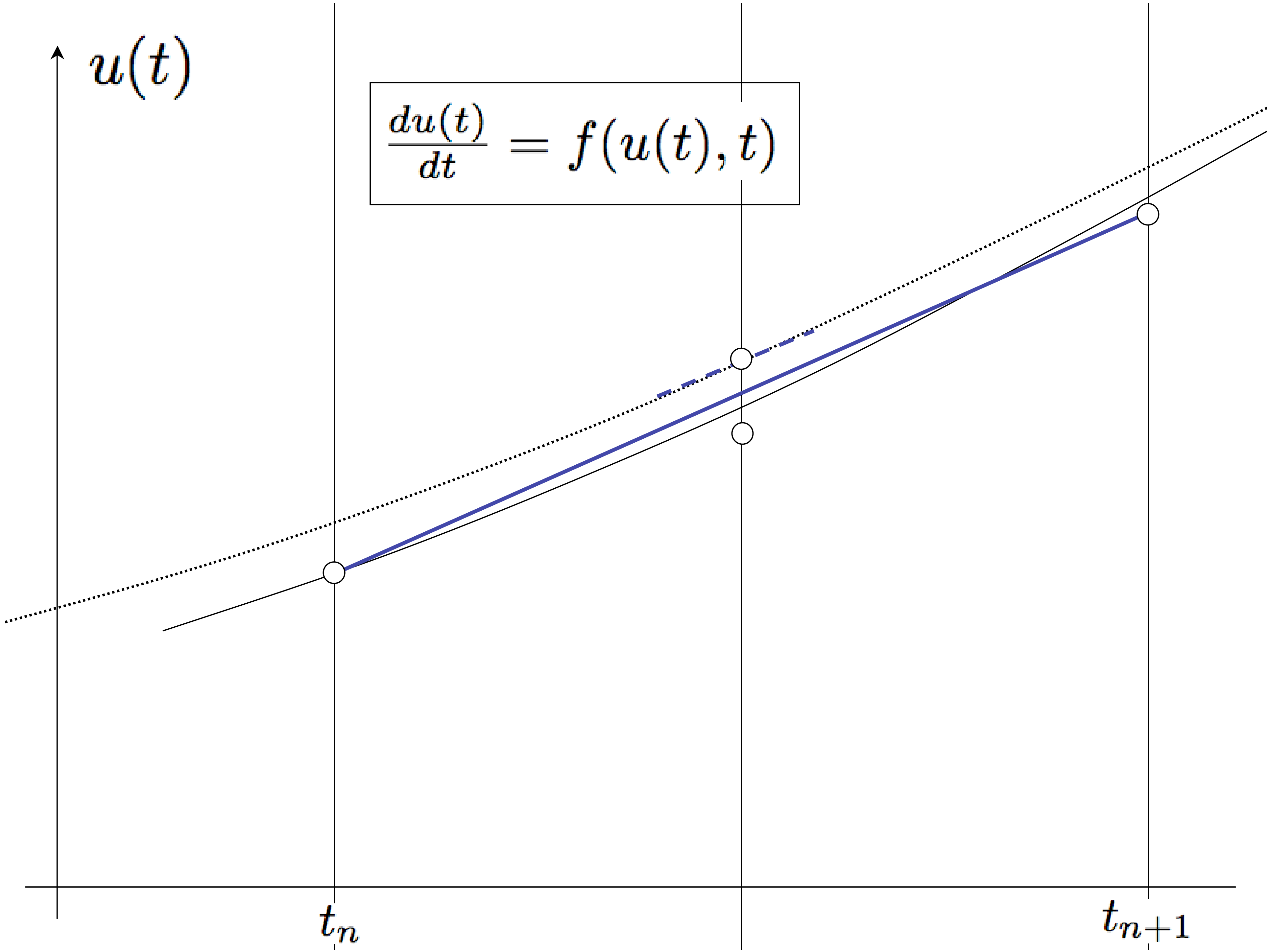
$u(t)$
$\frac{du(t)}{dt} = f(u(t), t)$
t_n
t_{n+1}



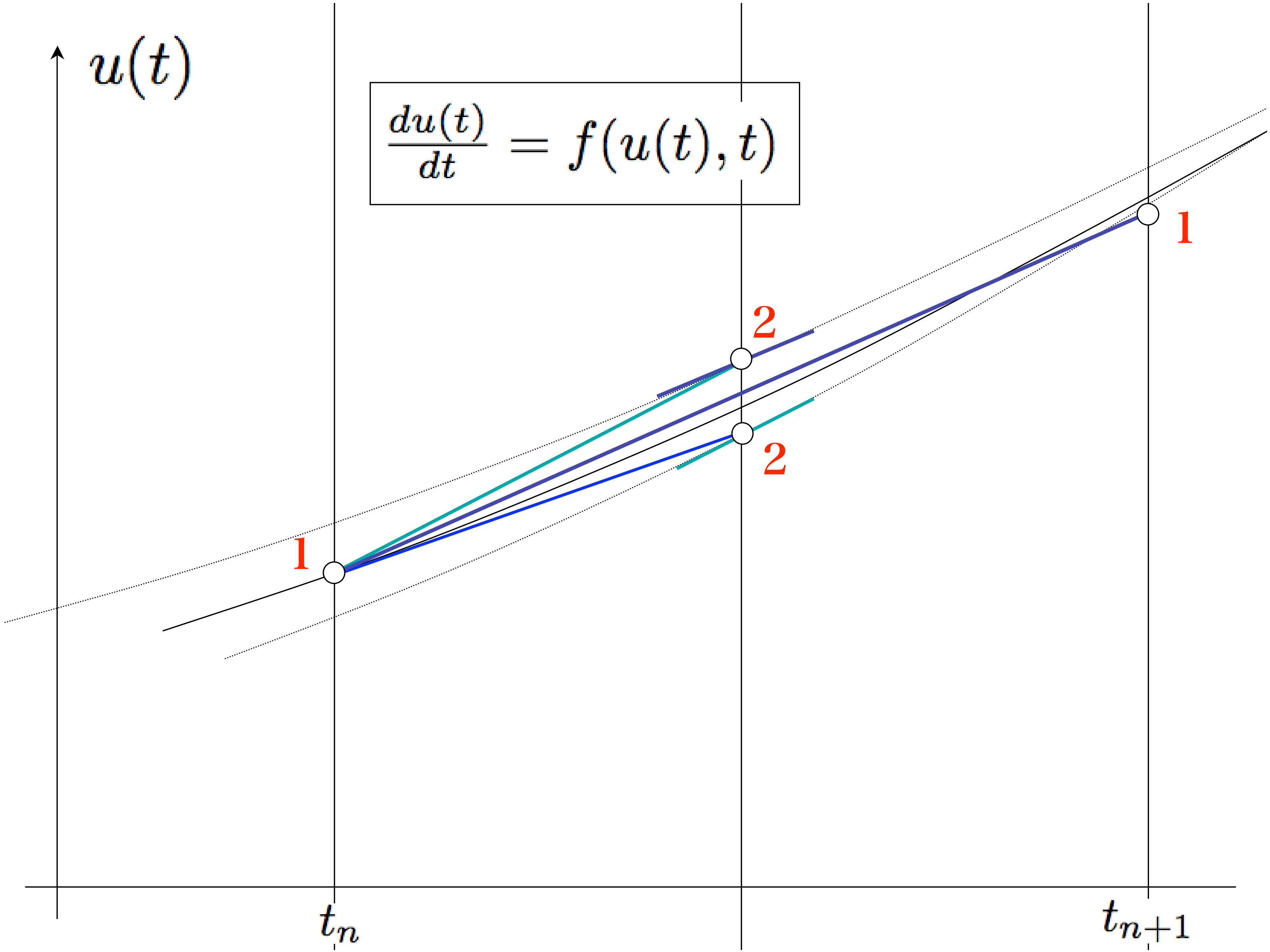
$u(t)$
$\frac{du(t)}{dt} = f(u(t), t)$
t_n
t_{n+1}



$u(t)$
$\frac{du(t)}{dt} = f(u(t), t)$
t_n
t_{n+1}



$u(t)$
$\frac{du(t)}{dt} = f(u(t), t)$
t_n
t_{n+1}



$u(t)$
$\frac{du(t)}{dt} = f(u(t), t)$
1
2
2
1
t_n
t_{n+1}

4-step 4th-order Runge-Kutta Integration Method

$$\frac{du(t)}{dt} = f(t, u(t))$$

$$t_0 = t_n$$
$$u_0 = u(t_0)$$
$$df_1 = \Delta t\, f(t_0, u_0)$$

$$t_1 = t_0 + 0.5\,\Delta t$$
$$u_1 = u_0 + 0.5\, df_1$$
$$df_2 = \Delta t\, f(t_1, u_1)$$

$$t_2 = t_0 + 0.5\,\Delta t$$
$$u_2 = u_0 + 0.5\, df_2$$
$$df_3 = \Delta t\, f(t_2, u_2)$$

$$t_3 = t_0 + \Delta t$$
$$u_3 = u_0 + df_3$$
$$df_4 = f(t_3, u_3)$$

$$u_{n+1} = u_n + \tfrac{1}{6}(df_1 + 2\, df_2 + 2\, df_3 + df_4)$$

Error $O(\Delta t^5)$ for one step, $O(\Delta t^4)$ in total.

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2}$$

Combination of RK + FDM

$$u_0 = e^{ikx_j}$$

⇐ A test wave

$$df_1 = \frac{\kappa \Delta t}{(\Delta x)^2} \left\{ e^{ik(x_j + \Delta x)} - 2\, e^{ikx_j} + e^{ik(x_j - \Delta x)} \right\}$$

$$= -\frac{2\kappa \Delta t}{(\Delta x)^2} (1 - \cos k\Delta x)\, e^{ikx_j}$$

$$= -\alpha\, e^{ikx_j}$$

$$\alpha = \frac{2\kappa \Delta t}{(\Delta x)^2} (1 - \cos k\Delta x)$$

$$u_1 = u_0 + 0.5\, df_1$$

$$= \left(1 - \tfrac{\alpha}{2}\right) e^{ikx_j}$$

$$df_2 = -\frac{2\, \kappa \Delta t}{(\Delta x)^2} \left(1 - \tfrac{\alpha}{2}\right) (1 - \cos k\Delta x)\, e^{ikx_j}$$

$$= -\alpha \left(1 - \tfrac{\alpha}{2}\right) e^{ikx_j}$$

$$
\begin{aligned}
u_3 &= u_0 + 0.5\, df_2 \\
&= \left(1 - \tfrac{\alpha}{2} + \tfrac{\alpha^2}{4}\right) e^{ikx_j} \\
df_3 &= -\left(\alpha - \tfrac{\alpha^2}{2} + \tfrac{\alpha^3}{4}\right) e^{ikx_j} \\
u_4 &= u_0 + df_3 \\
&= \left(1 - \alpha + \tfrac{\alpha^2}{2} - \tfrac{\alpha^3}{4}\right) e^{ikx_j} \\
df_4 &= -\left(\alpha - \alpha^2 + \tfrac{\alpha^3}{2} - \tfrac{\alpha^4}{4}\right) e^{ikx_j} \\
u_{\mathrm{new}} &= u_0 + \tfrac{1}{6}\left(df_1 + 2\, df_2 + 2\, df_3 + df_4\right) \\
&= \left(1 - \alpha + \tfrac{\alpha^2}{2} - \tfrac{\alpha^3}{6} + \tfrac{\alpha^4}{24}\right) e^{ikx_j} \\
&= \left\{1 + \frac{(-\alpha)}{1!} + \frac{(-\alpha)^2}{2!} + \frac{(-\alpha)^3}{3!} + \frac{(-\alpha)^4}{4!}\right\} e^{ikx_j}
\end{aligned}
$$

By one step integration of 4-th order Runge-Kutta method,

$$u_{\text{new}} = \left\{1 + \frac{(-\alpha)}{1!} + \frac{(-\alpha)^2}{2!} + \frac{(-\alpha)^3}{3!} + \frac{(-\alpha)^4}{4!}\right\} e^{ikx_j}$$

$$\alpha = \frac{2\kappa\Delta t}{(\Delta x)^2}(1 - \cos k\Delta x)$$

When $k\Delta x << 1$ $\quad \alpha \sim \frac{\kappa\Delta t}{(\Delta x)^2}(k\Delta x)^2 = k^2\,\kappa\Delta t$

$$u_{\text{exact}} = e^{-k^2\,\kappa\Delta t}\,e^{ikx_j} = e^{-\alpha}e^{ikx_j}$$

$\text{Error in 1step} = O[(\Delta t)^5]$ ⇒ $\text{Error in total} = O[(\Delta t)^4]$

By one step integration of 4-th order Runge-Kutta method,

$$u_{\mathrm{new}} = \left\{ 1 + \frac{(-\alpha)}{1!} + \frac{(-\alpha)^2}{2!} + \frac{(-\alpha)^3}{3!} + \frac{(-\alpha)^4}{4!} \right\} e^{ikx_j}$$

$$\alpha = \frac{2\kappa\Delta t}{(\Delta x)^2}(1 - \cos k\Delta x)$$

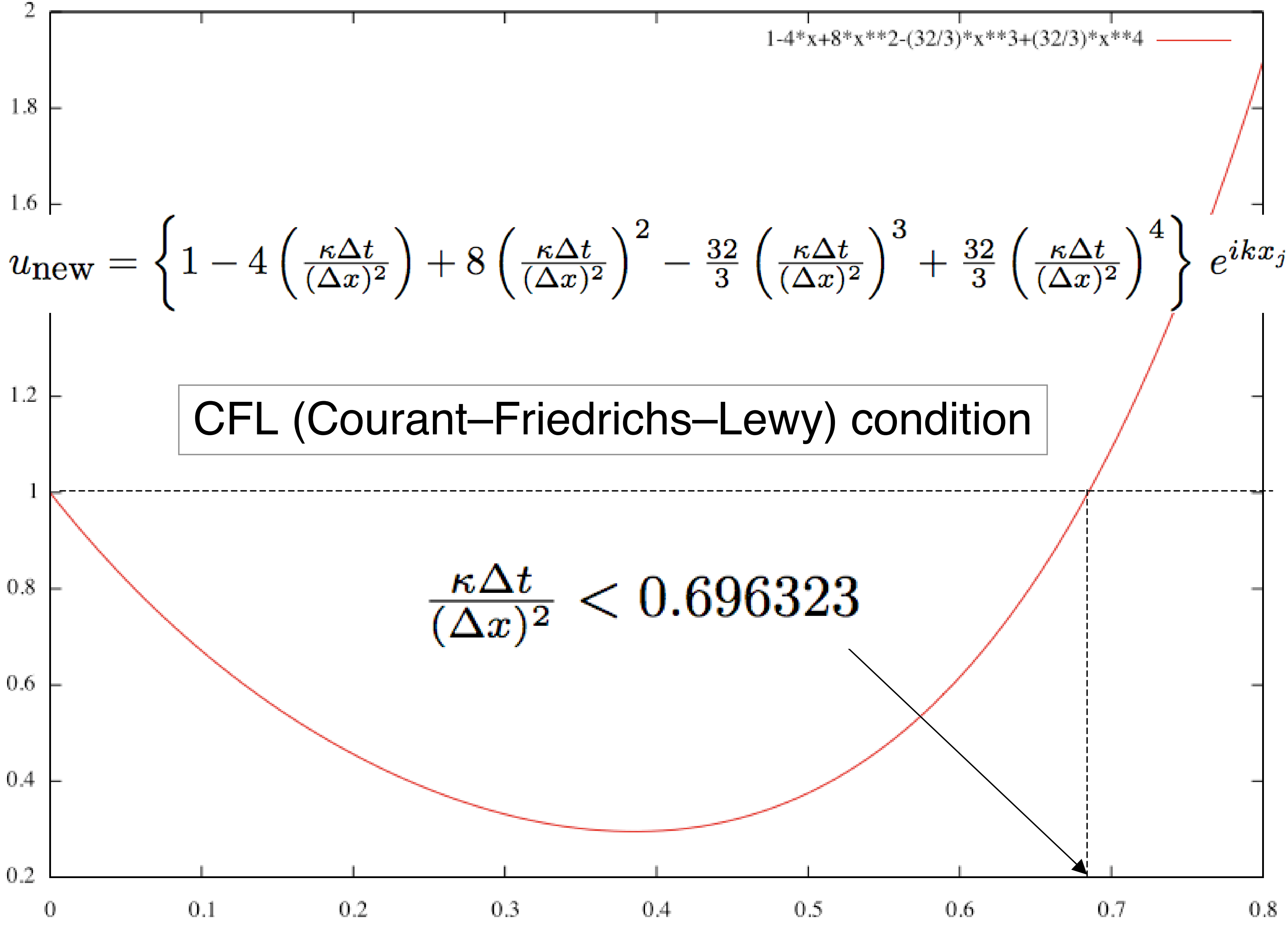
When $k\Delta x = \pi, \quad \alpha = \frac{4\kappa\Delta t}{(\Delta x)^2}$

$$u_{\mathrm{new}} = \left\{ 1 - 4\left(\frac{\kappa\Delta t}{(\Delta x)^2}\right) + 8\left(\frac{\kappa\Delta t}{(\Delta x)^2}\right)^2 - \frac{32}{3}\left(\frac{\kappa\Delta t}{(\Delta x)^2}\right)^3 + \frac{32}{3}\left(\frac{\kappa\Delta t}{(\Delta x)^2}\right)^4 \right\} e^{ikx_j}$$

$$u_{\text{new}} = \left\{ 1 - 4\left(\frac{\kappa \Delta t}{(\Delta x)^2}\right) + 8\left(\frac{\kappa \Delta t}{(\Delta x)^2}\right)^2 - \frac{32}{3}\left(\frac{\kappa \Delta t}{(\Delta x)^2}\right)^3 + \frac{32}{3}\left(\frac{\kappa \Delta t}{(\Delta x)^2}\right)^4 \right\} e^{ikx_j}$$

CFL (Courant–Friedrichs–Lewy) condition

$$\frac{\kappa \Delta t}{(\Delta x)^2} < 0.696323$$



Simple Numerical Simulation with Fortran90 Code

In sourcodes.tar.gz,
./src/DiffusionEquation/

$$\frac{\partial \psi}{\partial t} = \frac{\partial^2 \psi}{\partial x^2}$$

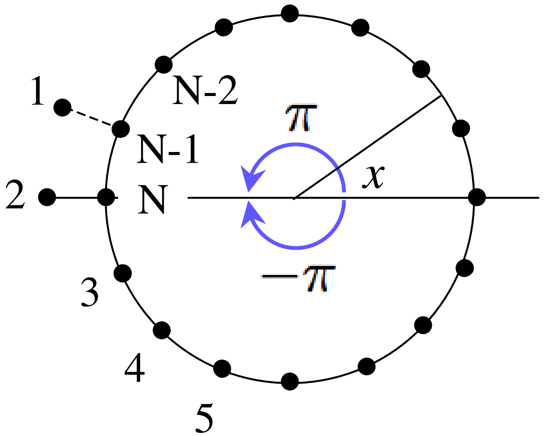
A Sample Code in FDM

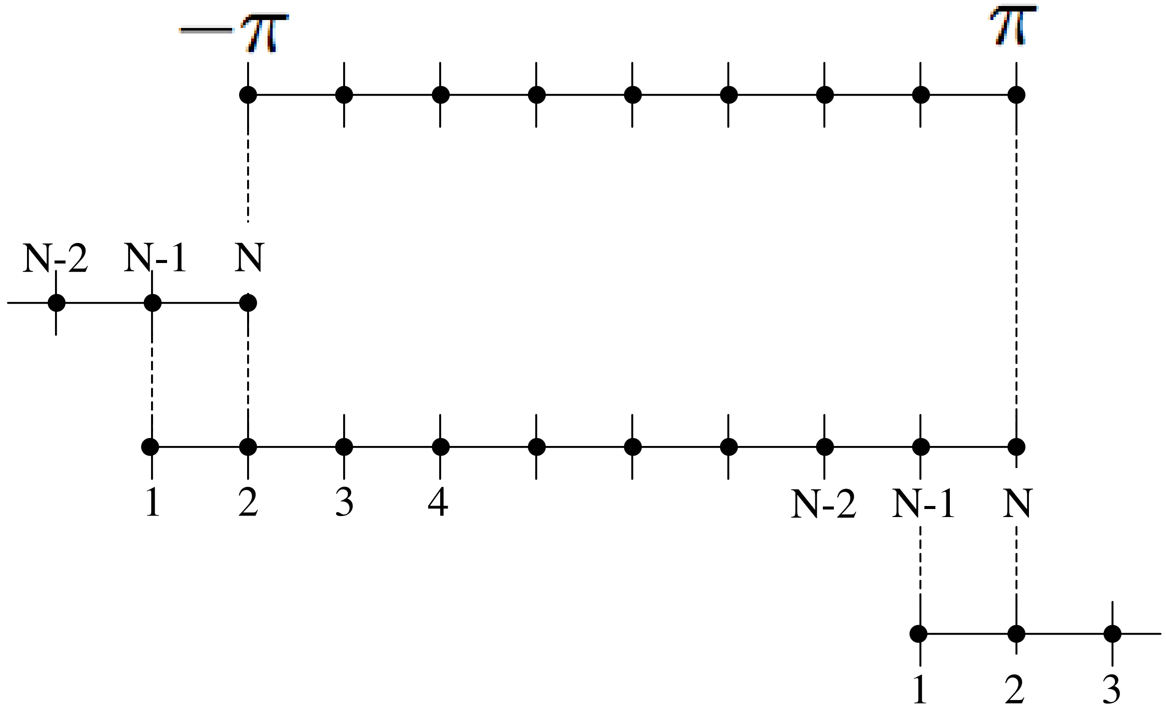
$$\frac{d\psi_j}{dt} = \frac{\psi_{j+1} - 2\psi_j + \psi_{j-1}}{(\Delta x)^2}$$

$$\frac{d\psi_j}{dt} = f(\psi_1, \psi_2, \cdots, \psi_N)$$

==> 4-step (4th order) Runge-Kutta method

Periodic boundary condition






```
subroutine iBoundary_condition(psi)
   real(DP), dimension(nx), intent(inout) :: psi

   psi(1)     = psi(nx-1)
   psi(nx)    = psi(2)

 end subroutine iBoundary_condition
```

Now let's see the code: main.f90

```
do nloop = 1 , nloop_max

   dpsi01(:) = rk4__step('1st',dt,dx,psi)
   call iBoundary_condition(dpsi01)

   dpsi02(:) = rk4__step('2nd',dt,dx,psi,dpsi01)
   call iBoundary_condition(dpsi02)

   dpsi03(:) = rk4__step('3rd',dt,dx,psi,dpsi02)
   call iBoundary_condition(dpsi03)

   dpsi04(:) = rk4__step('4th',dt,dx,psi,dpsi03)
   call iBoundary_condition(dpsi04)

   time = time + dt
   psi(:) = psi(:) + ONE_SIXTH*(dpsi01(:)      &
                            +2*dpsi02(:)      &
                            +2*dpsi03(:)      &
                              +dpsi04(:))

end do
```

Runge-Kutta step (rk.f90)

```
function rk4__step(nth,dt,dx,psi,dpsi_prev)      &
                                     result(dpsi_new)

    character(len=3), intent(in)          :: nth
    real(DP), intent(in)                  :: dt
    real(DP), intent(in)                  :: dx
    real(DP), dimension(:), intent(in)    :: psi
    real(DP), dimension(size(psi,dim=1)),    &
                   intent(in), optional :: dpsi_prev
    real(DP), dimension(size(psi,dim=1)) :: dpsi_new
    real(DP), dimension(size(psi,dim=1)) :: psi_
```

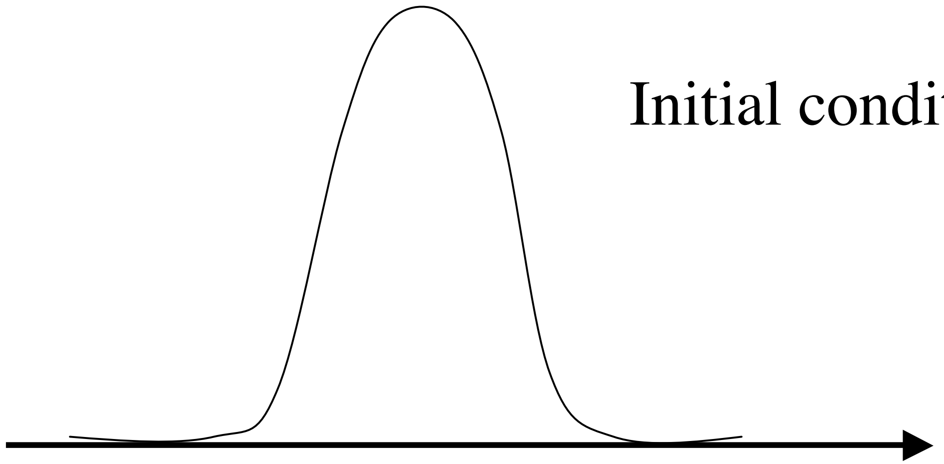
```fortran
  select case (nth)
  case ('1st')
    dpsi_new(:) = dt*diffusion_equation(size(psi,dim=1), &
                                        dx,psi)
  case ('2nd')
    psi_(:) = psi(:) + dpsi_prev(:)*0.5_DP
    dpsi_new(:) = dt*diffusion_equation(size(psi,dim=1), &
                                        dx,psi_)
  case ('3rd')
    psi_(:) = psi(:) + dpsi_prev(:)*0.5_DP
    dpsi_new(:) = dt*diffusion_equation(size(psi,dim=1), &
                                        dx,psi_)
  case ('4th')
    psi_(:) = psi(:) + dpsi_prev(:)
    dpsi_new(:) = dt*diffusion_equation(size(psi,dim=1), &
                                        dx,psi_)
  end select

end function rk4__step
```

diffusion_equation (rk.f90)

```
function diffusion_equation(nx,dx,psi)
   integer, intent(in)                    :: nx
   real(DP), intent(in)                   :: dx
   real(DP), dimension(nx), intent(in)    :: psi
   real(DP), dimension(nx)                :: diffusion_equation

   integer :: i
   real(DP) :: dx2

   dx2 = namelist__double('Diffusion_coeff')/(dx**2)

   do i = 2 , nx-1
      diffusion_equation(i) = dx2*(psi(i+1)-2*psi(i)+psi(i-1))
   end do

 end function diffusion_equation
```

$$\kappa \frac{\psi_{j+1} - 2\psi_j + \psi_{j-1}}{(\Delta x)^2}$$

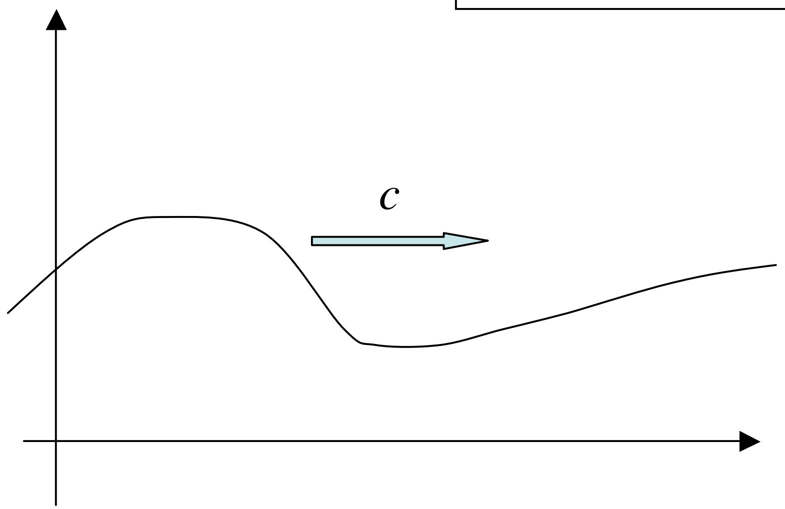
Let's run the code



Other equations by FDM (nonlinear terms)

Burgers' equation

$$\frac{\partial \psi}{\partial t} = -\psi \frac{\partial \psi}{\partial x} + \nu \frac{\partial^2 \psi}{\partial x^2}$$



$$\frac{\partial \psi}{\partial t} = -c \frac{\partial \psi}{\partial x}$$

Solution:

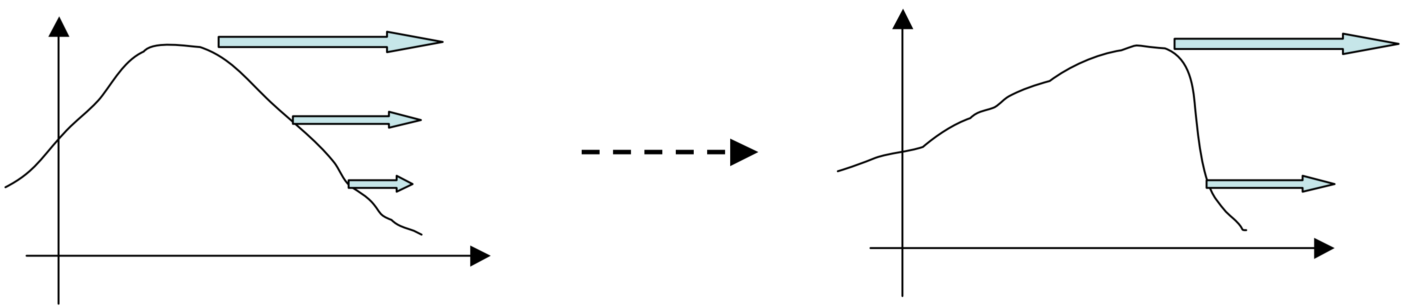
$$\psi(x, t) = f(x - ct)$$

Burgers' equation

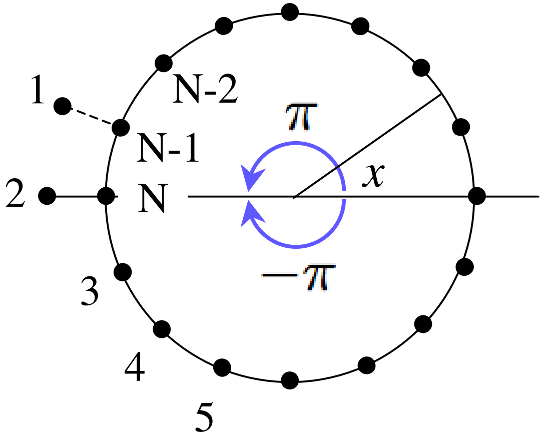
$$\frac{\partial \psi}{\partial t} = -\psi \frac{\partial \psi}{\partial x} + \nu \frac{\partial^2 \psi}{\partial x^2}$$

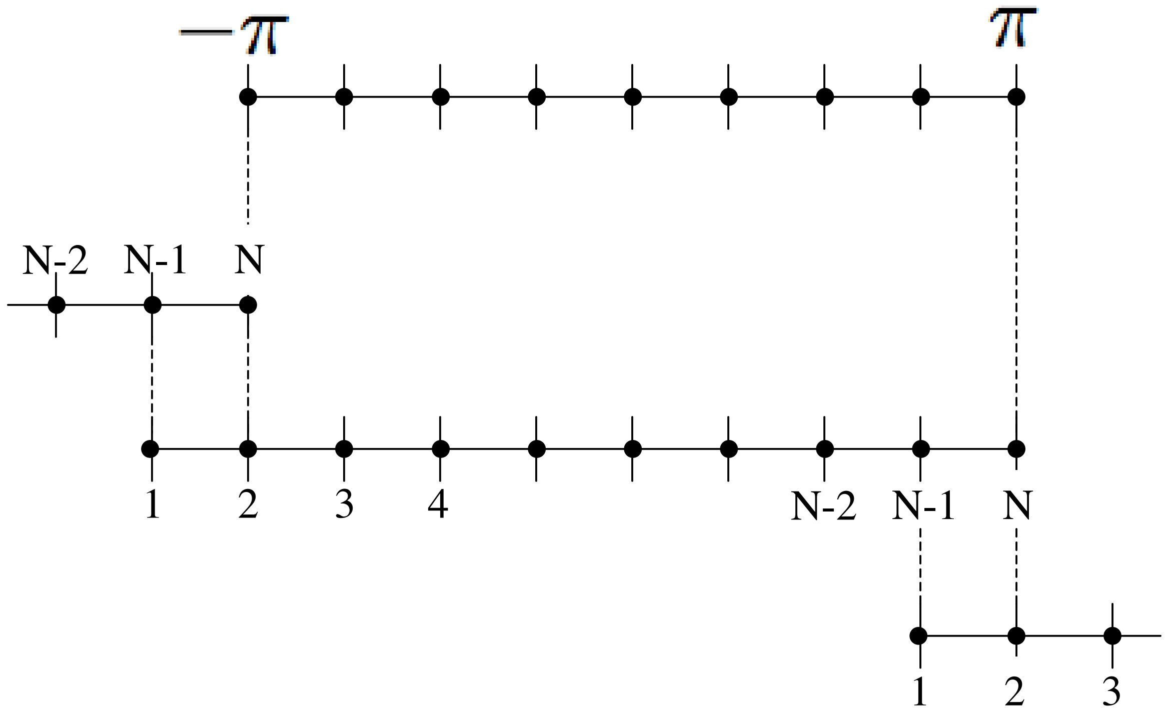
Diffusion term

$$\frac{\partial \psi}{\partial t} = -\psi \frac{\partial \psi}{\partial x}$$



Burgers' equation by FDM





$$\frac{d\psi_j}{dt} = -\psi_j \frac{\psi_{j+1} - \psi_{j-1}}{2\Delta x} + \nu \frac{\psi_{j+1} - 2\psi_j + \psi_{j-1}}{(\Delta x)^2}$$

Runge-Kutta 1st step (rk4.f90)

```
select case (nth)
   case ('1st')
      dpsi_new(:) = dt*burgers_equation(size(psi,dim=1),dx,psi)
   case ('2nd')
      psi_(:) = psi(:) + dpsi_prev(:)*0.5_DP
      dpsi_new(:) = dt*burgers_equation(size(psi,dim=1),dx,psi_)
   case ('3rd')
      psi_(:) = psi(:) + dpsi_prev(:)*0.5_DP
      dpsi_new(:) = dt*burgers_equation(size(psi,dim=1),dx,psi_)
   case ('4th')
      psi_(:) = psi(:) + dpsi_prev(:)
      dpsi_new(:) = dt*burgers_equation(size(psi,dim=1),dx,psi_)

end select
```

burgers_equation (rk4.f90)

```
function burgers_equation(nx,dx,psi)
  integer, intent(in)                          :: nx
  real(DP), intent(in)                         :: dx
  real(DP), dimension(nx), intent(in)          :: psi
  real(DP), dimension(nx)                      :: burgers_equation

  integer :: i
  real(DP) :: dx1, dx2

  dx1 = 1.0_DP / (2*dx)
  dx2 = namelist__double('Diffusion_coeff')/(dx**2)

  do i = 2 , nx-1
    burgers_equation(i) = - psi(i)*dx1*(psi(i+1)-psi(i-1)) &
                     + dx2*(psi(i+1)-2*psi(i)+psi(i-1))
  end do

end function burgers_equation
```

$$\frac{d\psi_j}{dt} = -\psi_j \frac{\psi_{j+1} - \psi_{j-1}}{2\Delta x} + \nu \frac{\psi_{j+1} - 2\psi_j + \psi_{j-1}}{(\Delta x)^2}$$

Let’s run the code