# MAINTITLE

Nikolaj Roager Christensen

Student Colloquium in Physics and Astronomy, Aarhus University

March 2021

TITLEIMAGE

# MAINTITLE

# Introduction, what and why

# Introduction, what and why

# Introduction, what and why

- 
  - 

# How: Numeric-ODE solvers

- When analytical solutions are not practical.

# How: Numeric-ODE solvers

- When analytical solutions are not practical.
- Testing experimental setups.

# How: Numeric-ODE solvers

- When analytical solutions are not practical.
- Testing experimental setups.

# How: Numeric-ODE solvers

- When analytical solutions are not practical.
- Testing experimental setups.

# How: Numeric-ODE solvers

- When analytical solutions are not practical.
- Testing experimental setups.
- Simulations are not experiments!

# Theory: Classical non-relativistic particles

- Some repetition from Electrodynamics

# Theory: Classical non-relativistic particles

- Some repetition from Electrodynamics
- The Lorentz force (SI units):

$$\vec{F} = q(\vec{v} \times \vec{B} + \vec{E}).$$

# Theory: Classical non-relativistic particles

- Some repetition from Electrodynamics
- The Lorentz force (SI units):

$$\vec{F} = q(\vec{v} \times \vec{B} + \vec{E}).$$

- Only 1 particle! so pre-programmed depending on the setup.

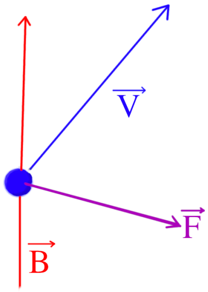
# Theory: Classical non-relativistic particles

- Some repetition from Electrodynamics
- The Lorentz force (SI units):

$$\vec{F} = q(\vec{v} \times \vec{B} + \vec{E}).$$

- Only 1 particle! so pre-programmed depending on the setup.
- Could use potentials $\phi(\vec{r}, t)$ $\vec{A}(\vec{r}, t)$ and Hamiltonian.

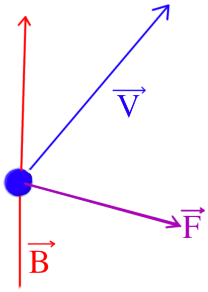
# Known results, cyclotron motion $\vec{B}$ fields



# Known results, cyclotron motion $\vec{B}$ fields

- Magnetic forces do no work:

$$dW_{\vec{B}} = \vec{F}_B \cdot d\vec{r} \propto (\vec{v} \times \vec{B}) \cdot \vec{v} = 0.$$



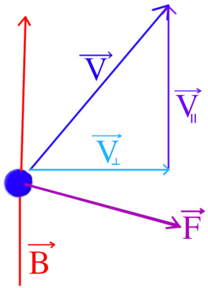
# Known results, cyclotron motion $\vec{B}$ fields

- Magnetic forces do no work:

$$dW_{\vec{B}} = \vec{F}_B \cdot d\vec{r} \propto (\vec{v} \times \vec{B}) \cdot \vec{v} = 0.$$

- $(\vec{v} = \vec{v}_\perp + \vec{v}_{||})$:

$$|\vec{F}_B| = |q(\vec{v} \times \vec{B})| = |qv_\perp B|.$$



# Known results, cyclotron motion $\vec{B}$ fields

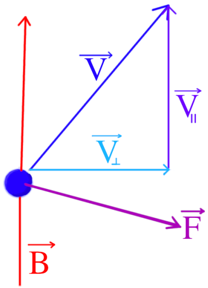
- Magnetic forces do no work:

$$dW_{\vec{B}} = \vec{F}_B \cdot d\vec{r} \propto (\vec{v} \times \vec{B}) \cdot \vec{v} = 0.$$

- $(\vec{v} = \vec{v}_\perp + \vec{v}_\|)$:

$$|\vec{F}_B| = |q(\vec{v} \times \vec{B})| = |qv_\perp B|.$$

- Same as Centripetal force: Cyclotron motion



# Known results, cyclotron motion $\vec{B}$ fields

- Magnetic forces do no work:

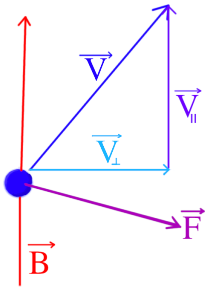
$$dW_{\vec{B}} = \vec{F}_B \cdot d\vec{r} \propto (\vec{v} \times \vec{B}) \cdot \vec{v} = 0.$$

- $(\vec{v} = \vec{v}_\perp + \vec{v}_\parallel)$:

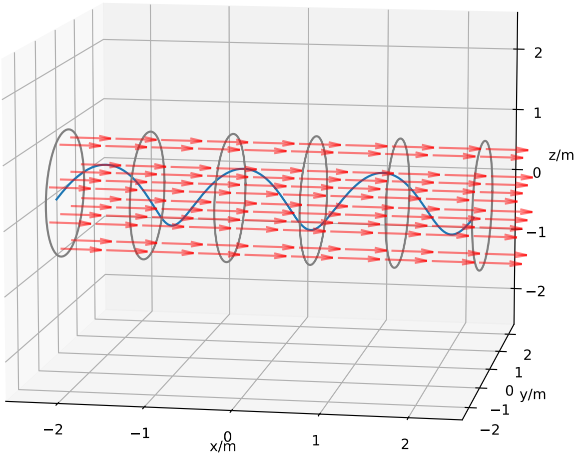
$$|\vec{F}_B| = |q(\vec{v} \times \vec{B})| = |qv_\perp B|.$$

- Same as Centripetal force: Cyclotron motion
- Cyclotron radius and frequency:

$$R = \frac{v_\perp m}{|q|B} \quad \omega_c = \frac{|q|B}{m}.$$

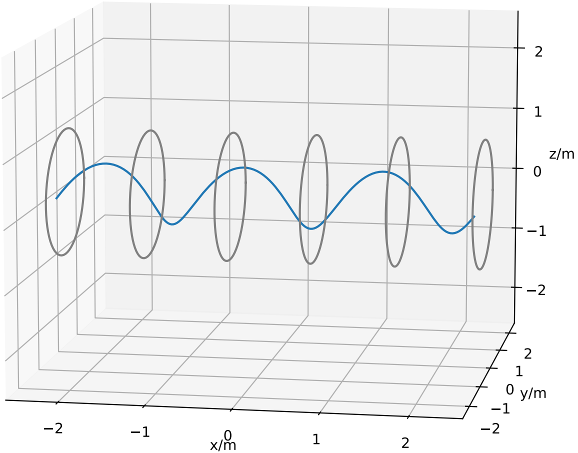


# Analytical solution: Protons in a Solenoid



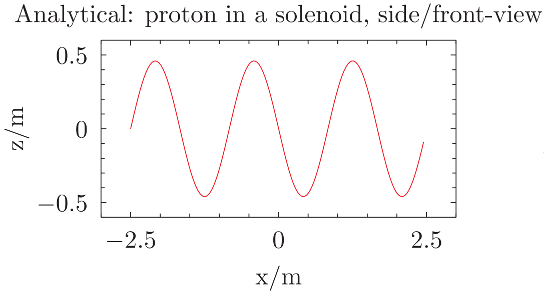

Solenoid with $N = 1000$ turns per $m$, $I = 5\,\mathrm{A}$, $r = 1\,\mathrm{m}$, $|\vec{B}| \approx 6\,\mathrm{mT}$.
Proton with $E_{kin} = 1\,\mathrm{MeV/c^2}$ ($|v| \approx 3.195 \times 10^5\,\mathrm{m/s}$)

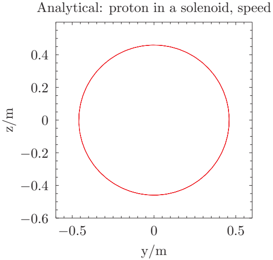
# Analytical solution: Protons in a Solenoid




$$R \approx 0.5\,\mathrm{m}\sin(\theta) \qquad T = \frac{2\pi}{\omega_c} \approx 10\,\mu\mathrm{s}$$

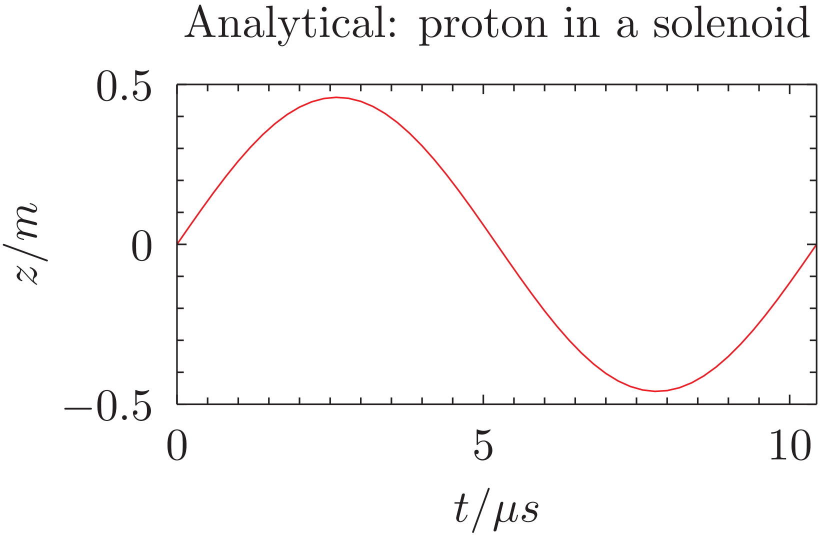
# Analytical solution: Protons in a Solenoid



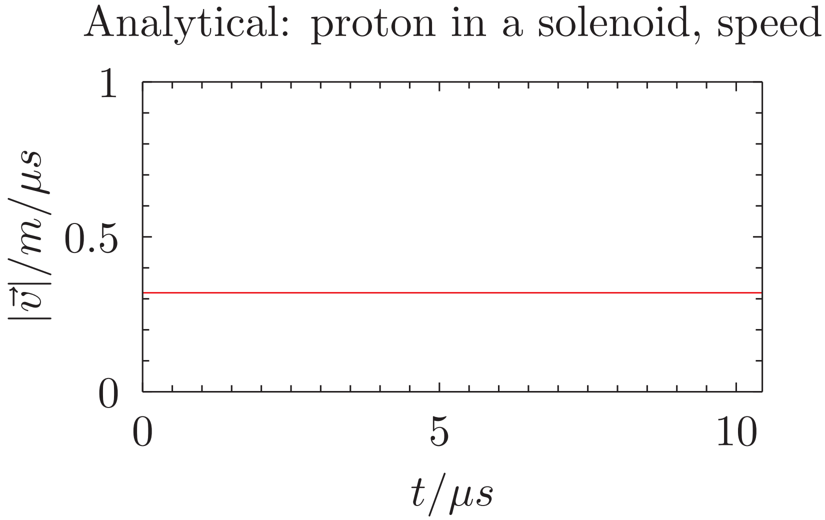

$$R \approx 0.5\,\mathrm{m}\sin(\theta) \qquad T = \frac{2\pi}{\omega_c} \approx 10\,\mu\mathrm{s}$$

# Analytical solution: Protons in a Solenoid




$$R \approx 0.5\,\mathrm{m}\sin(\theta) \qquad T = \frac{2\pi}{\omega_c} \approx 10\,\mu\mathrm{s}$$

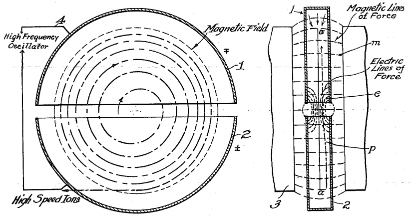
# Analytical solution: Protons in a Solenoid




$$R \approx 0.5\,\mathrm{m}\sin(\theta) \qquad T = \frac{2\pi}{\omega_c} \approx 10\,\mu\mathrm{s}$$

# Cyclotron accelerator

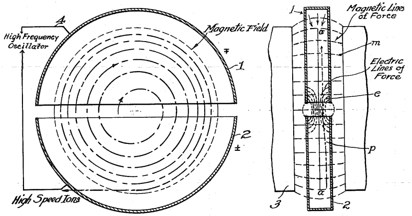
- Electric forces do work.




Ernest O. Lawrence, 1934, U.S. Patent 1,948,384; image in Public Domain.

# Cyclotron accelerator

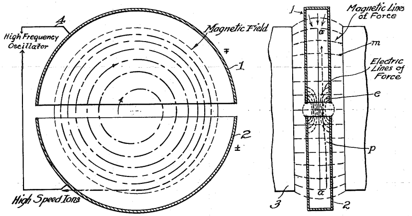
- Electric forces do work.
- Practical example, the Cyclotron.




Ernest O. Lawrence, 1934, U.S. Patent 1,948,384; image in Public Domain.

# Cyclotron accelerator

- Electric forces do work.
- Practical example, the Cyclotron.
- Single gab, oscillating field.

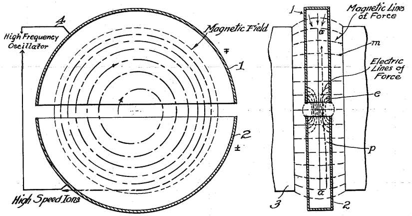



Ernest O. Lawrence, 1934, U.S. Patent 1,948,384; image in Public Domain.

# Cyclotron accelerator

- Electric forces do work.
- Practical example, the Cyclotron.
- Single gab, oscillating field.
- Final speed:

$$\frac{R|q|B}{m} = v_{\perp}$$




Ernest O. Lawrence, 1934, U.S. Patent 1,948,384; image in Public Domain.

# Ordinary differential equation*s.

- Sources: Zeigler et al. Theory of Modeling and Simulation (Third edition) chapter 3
- Algorithms exists for ODEs:

$$\dot{\mathbf{X}} = f_{ode}(\mathbf{X}(t), t).$$

# Ordinary differential equation*s.

- Sources: Zeigler et al. Theory of Modeling and Simulation (Third edition) chapter 3
- Algorithms exists for ODEs:

$$\dot{\mathbf{X}} = f_{ode}(\mathbf{X}(t), t).$$

- We have a:

$$\ddot{\vec{r}} = \frac{q}{m}(\dot{\vec{r}} \times \vec{B}(\vec{r}, t) + \vec{E}(\vec{r}, t)).$$

# Ordinary differential equation*s.

- Sources: Zeigler et al. Theory of Modeling and Simulation (Third edition) chapter 3
- Algorithms exists for ODEs:

$$\dot{\mathbf{X}} = f_{ode}(\mathbf{X}(t), t).$$

- We have a:

$$\ddot{\vec{r}} = \frac{q}{m}(\dot{\vec{r}} \times \vec{B}(\vec{r}, t) + \vec{E}(\vec{r}, t)).$$

- Here:

$$\mathbf{X} = \begin{pmatrix} \vec{r} \\ \dot{\vec{r}} \end{pmatrix} \quad f_{ode}(\vec{r}, t) = \begin{pmatrix} \dot{\vec{r}} \\ \frac{q}{m}(\dot{\vec{r}} \times \vec{B}(\vec{r}, t) + \vec{E}(\vec{r}, t)) \end{pmatrix}.$$

# The ODE to solve

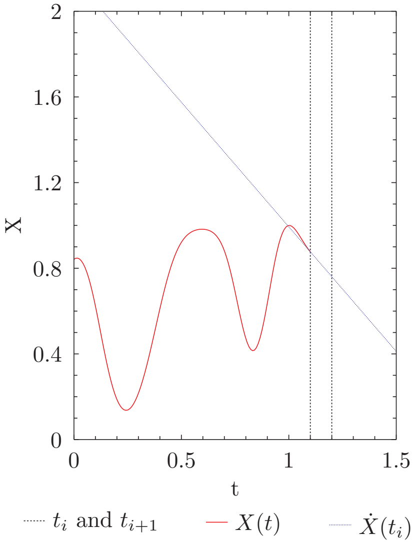
```
auto ODE = [...](const state_type Data, state_type &
    dDatadt, const double t){
     //Extract position and velocity from data
     vec pos = vec(Data[0],Data[1],Data[2]);
     vec velocity = vec(Data[3],Data[4],Data[5]);

     //Lorentz+Newtons 2nd law
     vec F = Charge*(Fields.get_Efield(pos,t)+
         cross(velocity,Fields.get_Bfield(pos,t)));
     vec dVdt = F*Inv_mass;

     //Save derivative of data
     dDatadt[0]=velocity.x;
     ...
};
```

# The Forward Euler's Method

- Let $h = t_{i+1} - t_i > 0$ be constant.



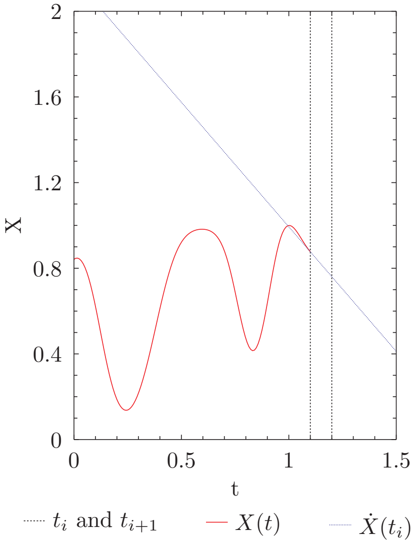

- Bernard P. Zeigler et al. Theory of Modeling and Simulation (Third edition), chapter 3

# The Forward Euler's Method

- Let $h = t_{i+1} - t_i > 0$ be constant.
- $h$, $\mathbf{X}(t)$, $t_i$ and $f_{ode}$ are known.

$$\dot{\mathbf{X}} = f_{ode}(\mathbf{X}(t), t).$$

- Bernard P. Zeigler et al. Theory of Modeling and Simulation (Third edition), chapter 3

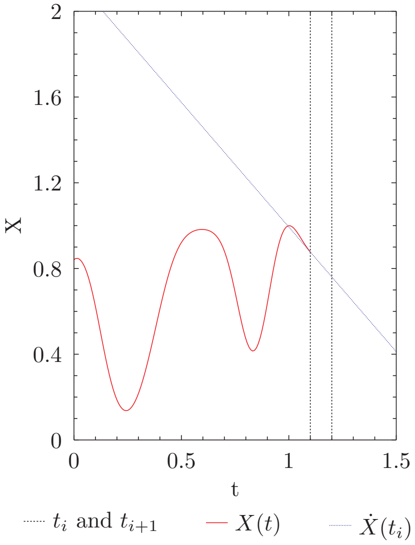


# The Forward Euler's Method

- Let $h = t_{i+1} - t_i > 0$ be constant.
- $h$, $\mathbf{X}(t)$, $t_i$ and $f_{ode}$ are known.

$$\dot{\mathbf{X}} = f_{ode}(\mathbf{X}(t), t).$$

- How would you find $\mathbf{X}(t_{i+1})$:




- Bernard P. Zeigler et al. Theory of Modeling and Simulation (Third edition), chapter 3

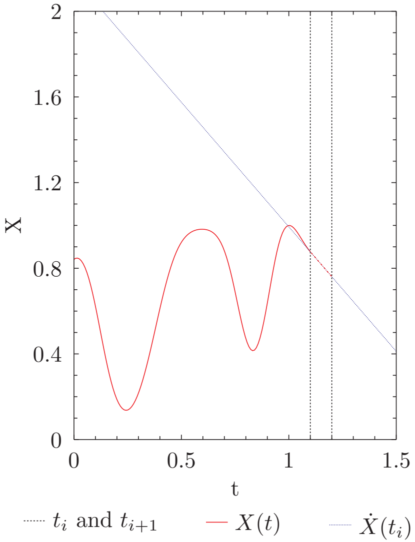
# The Forward Euler's Method

- Let $h = t_{i+1} - t_i > 0$ be constant.
- $h$, $\mathbf{X}(t)$, $t_i$ and $f_{ode}$ are known.

$$\dot{\mathbf{X}} = f_{ode}(\mathbf{X}(t), t).$$

- How would you find $\mathbf{X}(t_{i+1})$:

- Bernard P. Zeigler et al. Theory of Modeling and Simulation (Third edition), chapter 3



# The Forward Euler's Method

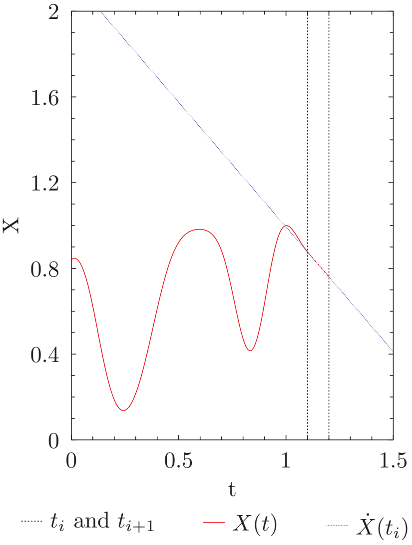
- Let $h = t_{i+1} - t_i > 0$ be constant.
- $h$, $\mathbf{X}(t)$, $t_i$ and $f_{ode}$ are known.

$$\dot{\mathbf{X}} = f_{ode}(\mathbf{X}(t), t).$$

- How would you find $\mathbf{X}(t_{i+1})$:
- (Explicit) Forward Euler's Method:

$$\mathbf{X}(t_{i+1}) = \mathbf{X}(t_i) + h f_{ode}(\mathbf{X}(t_i), t_i).$$

- Bernard P. Zeigler et al. Theory of Modeling and Simulation (Third edition), chapter 3



# The Forward Euler's Method

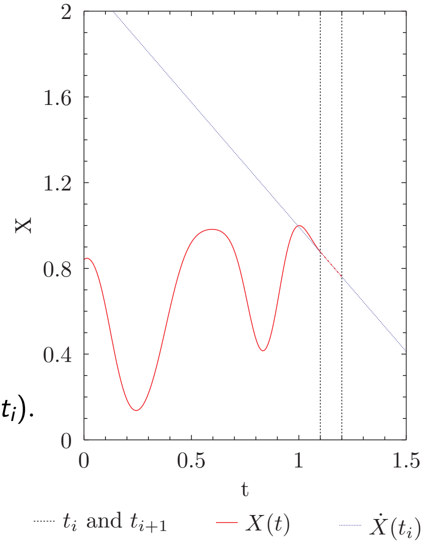
- Let $h = t_{i+1} - t_i > 0$ be constant.
- $h$, $\mathbf{X}(t)$, $t_i$ and $f_{ode}$ are known.

$$\dot{\mathbf{X}} = f_{ode}(\mathbf{X}(t), t).$$

- How would you find $\mathbf{X}(t_{i+1})$:
- (Implicit) Backward Euler's Method:

$$\mathbf{X}(t_{i+1}) = \mathbf{X}(t_i) + h f_{ode}(\mathbf{X}(t_{i+1}), t_i).$$

- Bernard P. Zeigler et al. Theory of Modeling and Simulation (Third edition), chapter 3



# The Forward Euler's Method

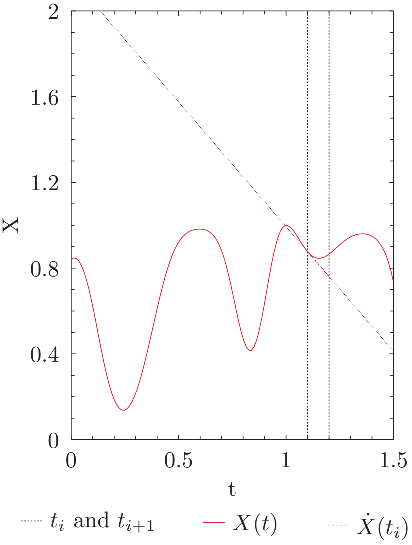
- Let $h = t_{i+1} - t_i > 0$ be constant.
- $h$, $\mathbf{X}(t)$, $t_i$ and $f_{ode}$ are known.

$$\dot{\mathbf{X}} = f_{ode}(\mathbf{X}(t), t).$$

- How would you find $\mathbf{X}(t_{i+1})$:
- (Explicit) Forward Euler's Method:

$$\mathbf{X}(t_{i+1}) = \mathbf{X}(t_i) + h f_{ode}(\mathbf{X}(t_i), t_i).$$

- Bernard P. Zeigler et al. Theory of Modeling and Simulation (Third edition), chapter 3



# Why does this work? What is the error

- Multiple justifications for why.

# Why does this work? What is the error

- Multiple justifications for why.
- First 2 terms in Taylor series Zeigler et al.:

$$\mathbf{X}(t_{i+1}) = \mathbf{X}(t_i) + hf_{ode}(\mathbf{X}(t_i), t_i) + h^2 \ldots + \ldots.$$

# Why does this work? What is the error

- Multiple justifications for why.
- First 2 terms in Taylor series Zeigler et al.:

$$\mathbf{X}(t_{i+1}) = \mathbf{X}(t_i) + hf_{ode}(\mathbf{X}(t_i), t_i) + h^2 \ldots + \ldots.$$

- "Local truncation error" $h^2 = h^{p+1}$.
- Global error $h = h^p$.

# Why does this work? What is the error

- Multiple justifications for why.
- First 2 terms in Taylor series Zeigler et al.:

$$\mathbf{X}(t_{i+1}) = \mathbf{X}(t_i) + hf_{ode}(\mathbf{X}(t_i), t_i) + h^2 \ldots + \ldots.$$

- "Local truncation error" $h^2 = h^{p+1}$.
- Global error $h = h^p$.
- Convergence, but not uniform.

# Why does this work? The Runge Kutta family

- In general.

$$\mathbf{X}(t_{i+1}) - \mathbf{X}(t_i) = \int_{t_i}^{t_{i+1}} f_{ode}(\mathbf{X}(t_i), t_i) dt h f_{ode}(\mathbf{X}(\tau), \tau)$$

- L. Zheng, X. Zhang, Modeling and Analysis of Modern Fluid Problems, 2017, chapter 8:

# Why does this work? The Runge Kutta family

- In general.

$$\mathbf{X}(t_{i+1}) - \mathbf{X}(t_i) = \int_{t_i}^{t_{i+1}} f_{ode}(\mathbf{X}(t_i), t_i)dt = hf_{ode}(\mathbf{X}(\tau), \tau)$$

- *Mean Value theorem for integrals* $t_i \leq \tau \leq t_{i+1}$.

- L. Zheng, X. Zhang, Modeling and Analysis of Modern Fluid Problems, 2017, chapter 8:

# Why does this work? The Runge Kutta family

- In general.

$$\mathbf{X}(t_{i+1}) - \mathbf{X}(t_i) = \int_{t_i}^{t_{i+1}} f_{ode}(\mathbf{X}(t_i), t_i)dt = hf_{ode}(\mathbf{X}(\tau), \tau)$$

- *Mean Value theorem for integrals* $t_i \leq \tau \leq t_{i+1}$.
- Guess $\tau = t_i$.

- L. Zheng, X. Zhang, Modeling and Analysis of Modern Fluid Problems, 2017, chapter 8:

# Why does this work? The Runge Kutta family

- In general.

$$\mathbf{X}(t_{i+1}) - \mathbf{X}(t_i) = \int_{t_i}^{t_{i+1}} f_{ode}(\mathbf{X}(t_i), t_i)dt = hf_{ode}(\mathbf{X}(\tau), \tau)$$

- *Mean Value theorem for integrals* $t_i \leq \tau \leq t_{i+1}$.
- Guess $\tau = t_i$.
- More generally, (*Explicit* and *single step*), Runge-Kutta family:

$$\mathbf{X}(t_{i+1}) - \mathbf{X}(t_i) = \int_{t_i}^{t'} f_{ode}(\mathbf{X}(t_i), t_i)dt + \dots \int_{t^{(m)}}^{t_{i+1}} f_{ode}(\mathbf{X}(t_i), t_i)dt$$

- L. Zheng, X. Zhang, Modeling and Analysis of Modern Fluid Problems, 2017, chapter 8:

# Why does this work? The Runge Kutta family

- In general.

$$\mathbf{X}(t_{i+1}) - \mathbf{X}(t_i) = \int_{t_i}^{t_{i+1}} f_{ode}(\mathbf{X}(t_i), t_i) dt = h f_{ode}(\mathbf{X}(\tau), \tau)$$

- *Mean Value theorem for integrals* $t_i \leq \tau \leq t_{i+1}$.
- Guess $\tau = t_i$.
- More generally, (*Explicit* and *single step*), Runge-Kutta family:

$$\begin{aligned}\mathbf{X}(t_{i+1}) - \mathbf{X}(t_i) &= \int_{t_i}^{t'} f_{ode}(\mathbf{X}(t_i), t_i) dt + \ldots \int_{t^{(m)}}^{t_{i+1}} f_{ode}(\mathbf{X}(t_i), t_i) dt \\ &= h \sum_{j=1}^{m} c_j f_{ode}(\mathbf{X}(\tau_j), \tau_j)\end{aligned}$$

- Use $f_{ode}(\mathbf{X}(t_i), t_i)$ to approximate $\mathbf{X}(\tau_1)$ etc.
- L. Zheng, X. Zhang, Modeling and Analysis of Modern Fluid Problems, 2017, chapter 8:

# Explicit Runge Kutta methods

- We want:

$$\mathbf{X}(t_{i+1}) - \mathbf{X}(t_i) = h \sum_{j=1}^{m} b_j \mathbf{K}_j$$

- With: $\mathbf{K}_1 = f_{ode}(\mathbf{X}(t_i), t_i)$, $\mathbf{K}_2 = f_{ode}(\mathbf{X}(t_i) + ha_{21}\mathbf{K}_1, t_i + c_2 h)$ etc.

- Martha L. Abell, James P. Braselton, Differential Equations with Mathematica (Fourth Edition), 2016:

# Explicit Runge Kutta methods

- We want:

$$\mathbf{X}(t_{i+1}) - \mathbf{X}(t_i) = h \sum_{j=1}^{m} b_j \mathbf{K}_j$$

- With: $\mathbf{K}_1 = f_{ode}(\mathbf{X}(t_i), t_i)$, $\mathbf{K}_2 = f_{ode}(\mathbf{X}(t_i) + ha_{21}\mathbf{K}_1, t_i + c_2 h)$ etc.
- Can be found with taylor expansion of $\mathbf{X}(t_i)$.


- Martha L. Abell, James P. Braselton, Differential Equations with Mathematica (Fourth Edition), 2016:

# Explicit Runge Kutta methods

- We want:

$$\mathbf{X}(t_{i+1}) - \mathbf{X}(t_i) = h \sum_{j=1}^{m} b_j \mathbf{K}_j$$

- With: $\mathbf{K}_1 = f_{ode}(\mathbf{X}(t_i), t_i)$, $\mathbf{K}_2 = f_{ode}(\mathbf{X}(t_i) + h a_{21}\mathbf{K}_1, t_i + c_2 h)$ etc.
- Can be found with taylor expansion of $\mathbf{X}(t_i)$.
- 2nd order (Heun's method):

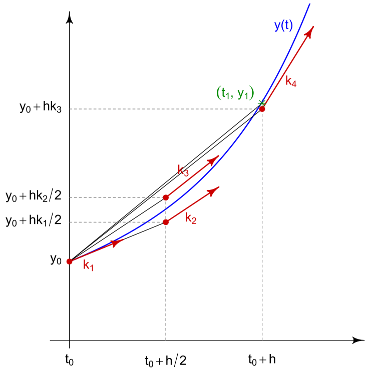
$$\begin{aligned}
\mathbf{X}(t_{i+1}) - \mathbf{X}(t_i) &= \frac{h}{2}(\mathbf{k}_1 + \mathbf{k}_2) \\
\mathbf{k}_1 &= f_{ode}(\mathbf{X}(t_i), t_i) \\
\mathbf{k}_2 &= f_{ode}(\mathbf{X}(t_i) + h\mathbf{k}_1, t_i + h)
\end{aligned}$$

- Martha L. Abell, James P. Braselton, Differential Equations with Mathematica (Fourth Edition), 2016:

# The 4th order Runge Kutta method

- RK4, often simply called the Runge Kutta method:

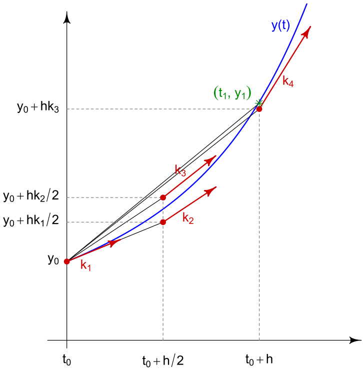
$$
\begin{aligned}
\mathbf{X}(t_{i+1}) - \mathbf{X}(t_i) &= \frac{h}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4) \\
\mathbf{k}_1 &= f_{ode}(\mathbf{X}(t_i), t_i) \\
\mathbf{k}_2 &= f_{ode}(\mathbf{X}(t_i) + \frac{h}{2}\mathbf{k}_1, t_i + \frac{h}{2}) \\
\mathbf{k}_3 &= f_{ode}(\mathbf{X}(t_i) + \frac{h}{2}\mathbf{k}_2, t_i + \frac{h}{2}) \\
\mathbf{k}_4 &= f_{ode}(\mathbf{X}(t_i) + h\mathbf{k}_3, t_i + h)
\end{aligned}
$$



# The 4th order Runge Kutta method

- RK4, often simply called the Runge Kutta method:

$$
\begin{aligned}
\mathbf{X}(t_{i+1}) - \mathbf{X}(t_i) &= \frac{h}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4) \\
\mathbf{k}_1 &= f_{ode}(\mathbf{X}(t_i), t_i) \\
\mathbf{k}_2 &= f_{ode}(\mathbf{X}(t_i) + \frac{h}{2}\mathbf{k}_1, t_i + \frac{h}{2}) \\
\mathbf{k}_3 &= f_{ode}(\mathbf{X}(t_i) + \frac{h}{2}\mathbf{k}_2, t_i + \frac{h}{2}) \\
\mathbf{k}_4 &= f_{ode}(\mathbf{X}(t_i) + h\mathbf{k}_3, t_i + h)
\end{aligned}
$$




Wikipedia-user HilberTraum, published under creative commins: CC BY-SA 4.0

- Almost default in `scipy.integrate.solve_ivp` and matlab `ode45`.

# Euler Implementations

[fragile]

```
state_type Data = Data0;
state_type dDatadt;
size_t time_res = T/timestep;
for (size_t i = 1; i < time_res; ++i)
{
    double t=i*dt;
    ODE(Data,dDatadt,t);
    //Euler time evolution
    //Data +=timestep*dDatadt; 1 variable
    for (uint i = 0; i<Data.size(); ++i)
        Data[i]+=timestep*dDatadt[i];
    save_step( Data , i*timestep );
};
```

# RK4 Implementations (1/2)

```
state_type Data = Data0;
state_type temp=Data0;
state_type K1,K2,K3,K4;
size_t time_res = T/timestep;
for (size_t i = 1; i < time_res; ++i)
{
    double t=i*timestep;
    //substep 1
    ODE(Data,K1,t);
    for (uint i = 0; i<Data.size(); ++i)
        temp[i]=Data[i]+timestep*K1[i]/2;
    //substep 2
    ODE(Data,K2,t+timestep/2);
    for (uint i = 0; i<Data.size(); ++i)
        temp[i]=Data[i]+timestep*K2[i]/2;
```

# RK4 Implementations (2/2)

```
    //substep 3
    ODE(Data,K3,t+timestep/2);
    for (uint i = 0; i<Data.size(); ++i)
        temp[i]=Data[i]+timestep*K3[i];
    //substep 4
    ODE(temp,K4,t+timestep);
    //Read data
    for (uint i = 0; i<Data.size(); ++i)
        Data[i]+=timestep*(K1[i]+2.0*K2[i]+2.0*K3[i]+
   K4[i])/6.0;
    save_step( Data , i*timestep );
}
```

# “Correct” way

```
#include <boost/array.hpp>
#include <boost/numeric/odeint.hpp>
using namespace boost::numeric::odeint;
typedef boost::array< double, 6 > state_type;
...
size_t steps = integrate_const(
    runge_kutta4< state_type >(),
    ODE,    //Lorentz-force
    Data0 ,//{pos0,v0}
    0.0 ,   //t0=0
    T ,     //max time
    timestep ,//length of each step
    save_step //User defined save data function
);
```

# Does it work

- Test, same proton in a solenoid use $\theta = 60^\circ$ reference, had:

$$R \approx 0.5\,\mathrm{m}\sin(\theta) \approx 0.45\,\mathrm{m} \quad T = \frac{2\pi}{\omega_c} \approx 10\,\mu\mathrm{s}$$

# Does it work

- Test, same proton in a solenoid use $\theta = 60^\circ$ reference, had:

$$R \approx 0.5\,\mathrm{m}\sin(\theta) \approx 0.45\,\mathrm{m} \quad T = \frac{2\pi}{\omega_c} \approx 10\,\mu\mathrm{s}$$

- Compare Analytic, Euler, Runge-Kutta 4.

# Does it work

- Test, same proton in a solenoid use $\theta = 60^\circ$ reference, had:

$$R \approx 0.5\,\mathrm{m}\sin(\theta) \approx 0.45\,\mathrm{m} \quad T = \frac{2\pi}{\omega_c} \approx 10\,\mu\mathrm{s}$$

- Compare Analytic, Euler, Runge-Kutta 4.
- Consider $\theta = 60^\circ$, $h = 0.01\,\mu\mathrm{s}$, $h = 0.1\,\mu\mathrm{s}$ and $h = 0.1\,\mu\mathrm{s}$.

# Does it work

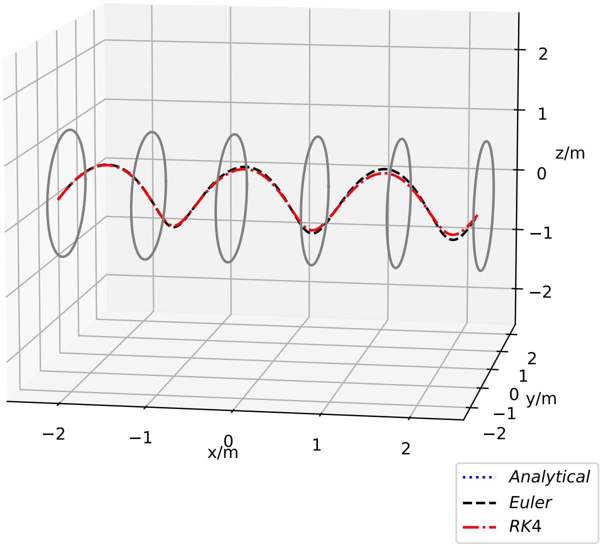
- Test, same proton in a solenoid use $\theta = 60^\circ$ reference, had:

$$R \approx 0.5\,\mathrm{m}\sin(\theta) \approx 0.45\,\mathrm{m} \quad T = \frac{2\pi}{\omega_c} \approx 10\,\mu\mathrm{s}$$

- Compare Analytic, Euler, Runge-Kutta 4.
- Consider $\theta = 60^\circ$, $h = 0.01\,\mu\mathrm{s}$, $h = 0.1\,\mu\mathrm{s}$ and $h = 0.1\,\mu\mathrm{s}$.
- Check error on $|\vec{v}|$, $R = \sqrt{y^2 + z^2}$ and $x(t)$.

# At a glance, 3D view

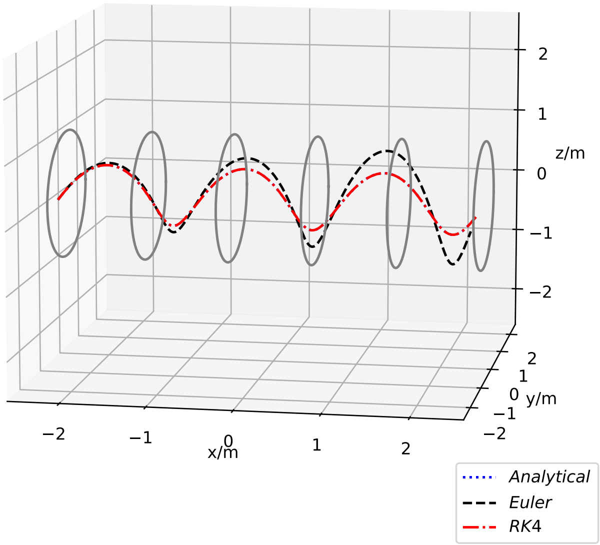
$h = t_{i+1} - t_i = 0.01\,\mu\text{s}$




3129 steps

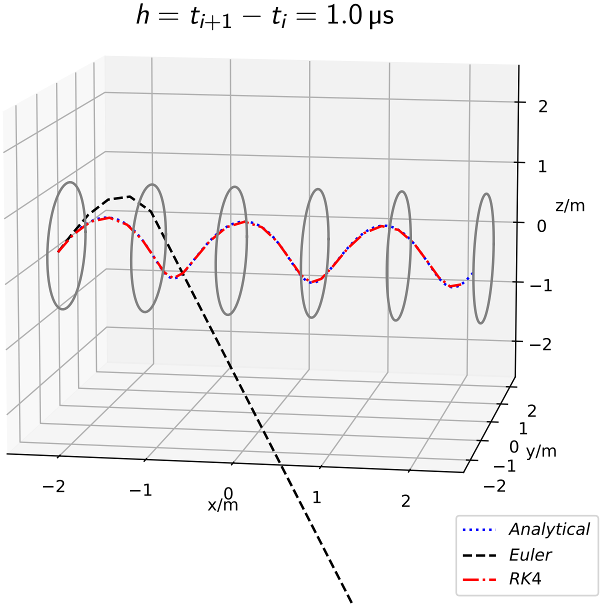
# At a glance, 3D view

$h = t_{i+1} - t_i = 0.1\,\mu s$



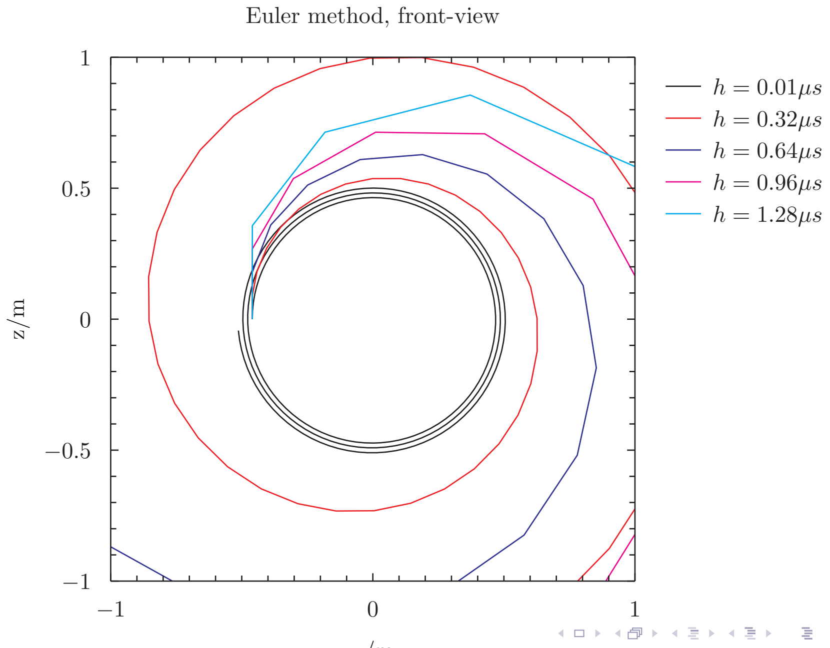

312 steps

# At a glance, 3D view

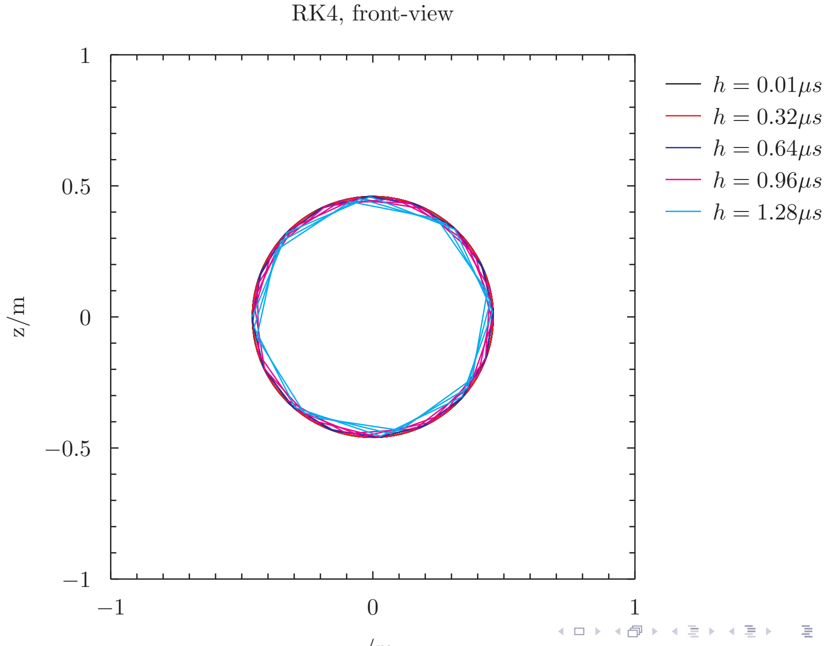



31 steps.

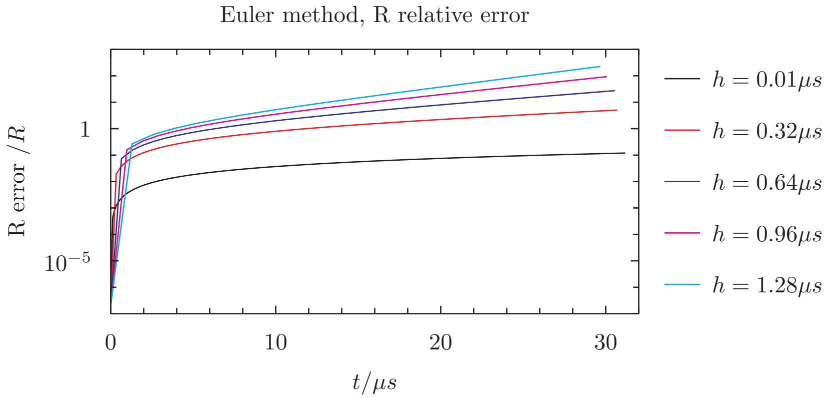
# At a glance, front view, no border



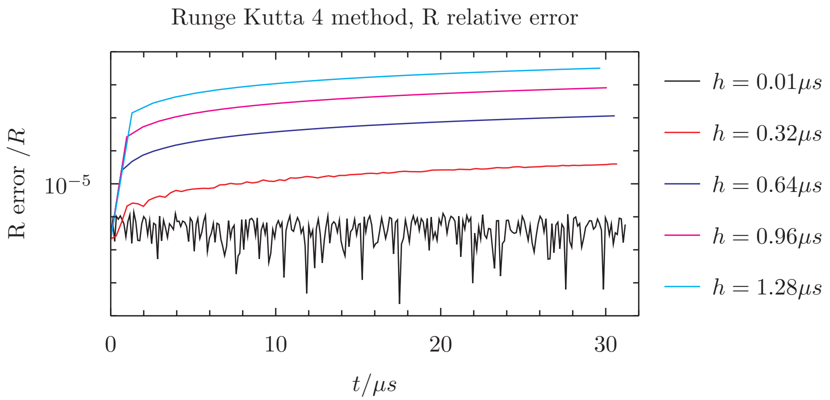
# At a glance, front view, no border



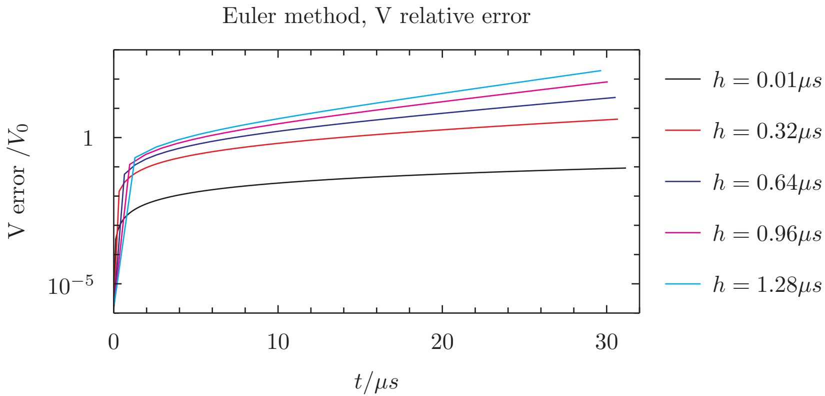
# Constant radius?



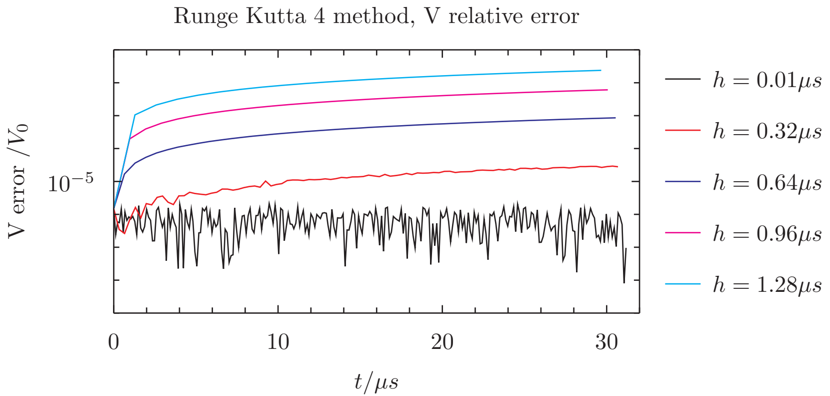
# Constant radius?



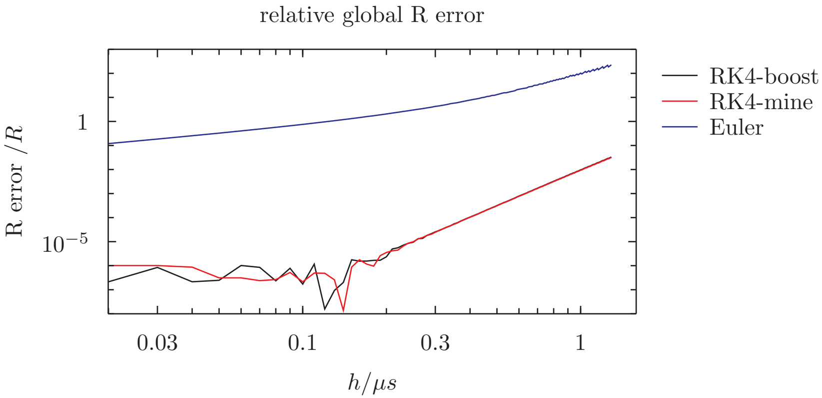
# Constant speed?



# Constant speed?



# Order of the error



# Adaptive step size, introduce it + when it fails

# Non-analytic systems

# Questions