How numerical simulations work: Simulating particles in electric and magnetic fields

Nikolaj Roager Christensen

Student Colloquium in Physics and Astronomy, Aarhus University

March 2021

TITLEIMAGE

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.

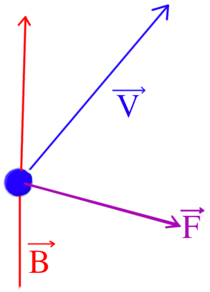
Theory: Classical non-relativistic particles

- 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.

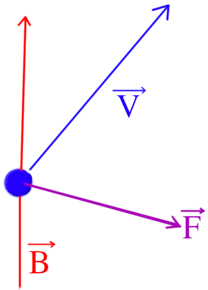
Known results, cyclotron motion $\vec{B}$ fields



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

- Magnetic forces do no work:

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



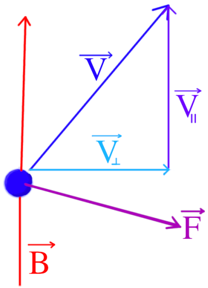
Known results, cyclotron motion $\vec{B}$ fields

- 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

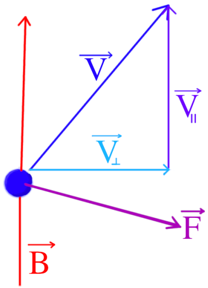
- Magnetic forces do no work:

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

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

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

- Same as Centripetal force: Cyclotron motion



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

- Magnetic forces do no work:

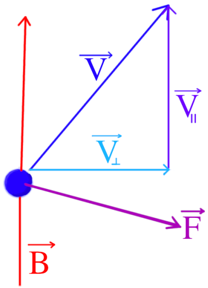
$$dW_{\vec{B}} = \vec{F}_B \cdot d\vec{r} \propto (\vec{v} \times \vec{B}) \cdot \vec{v} = 0.$$

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

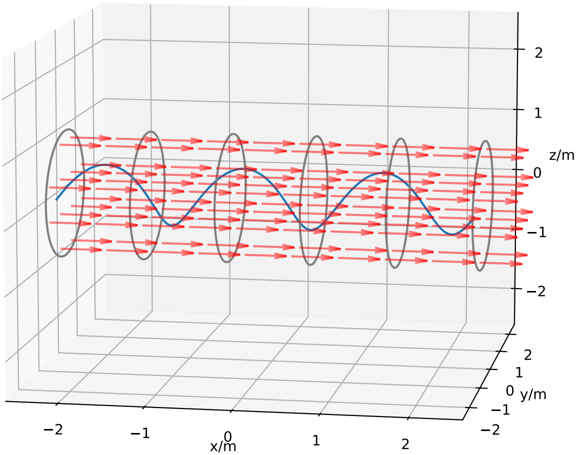
$$|\vec{F}_B| = |q(\vec{v} \times \vec{B})| = |q v_\perp B|.$$

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

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

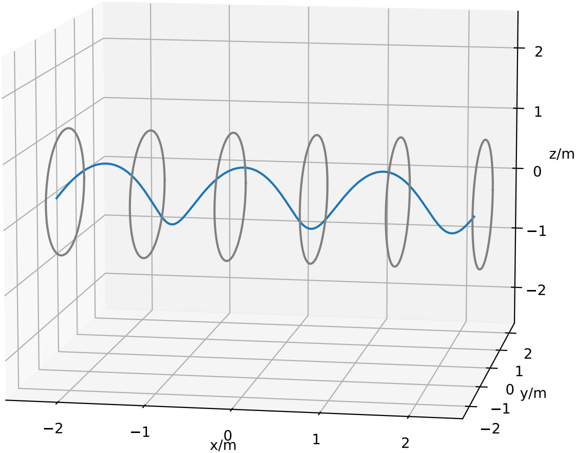


Analytical solution: Protons in a Solenoid



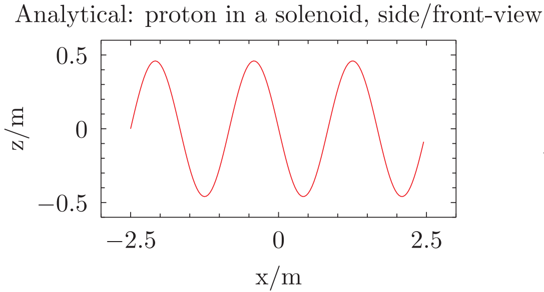
Solenoid with $N = 1000$ turns per m, $I = 5\,\mathrm{A}$, $r = 1\,\mathrm{m}$, $|\vec{B}| \approx 6\,\mathrm{mT}$.
Proton with $E_{kin} = 1\,\mathrm{MeV/c^2}$ ($|v| \approx 3.195 \times 10^5\,\mathrm{m/s}$)

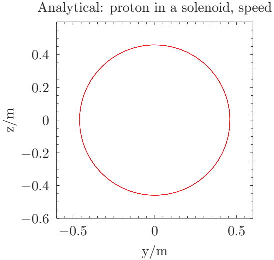
Analytical solution: Protons in a Solenoid



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

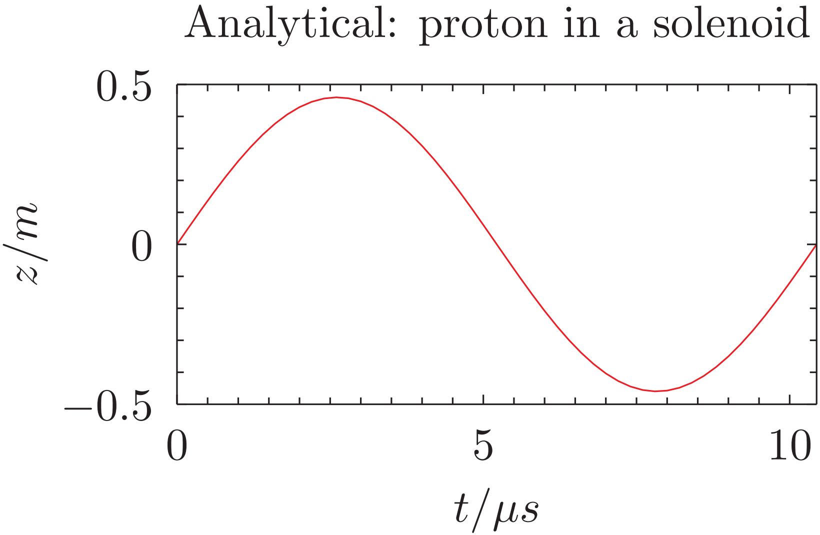
Analytical solution: Protons in a Solenoid





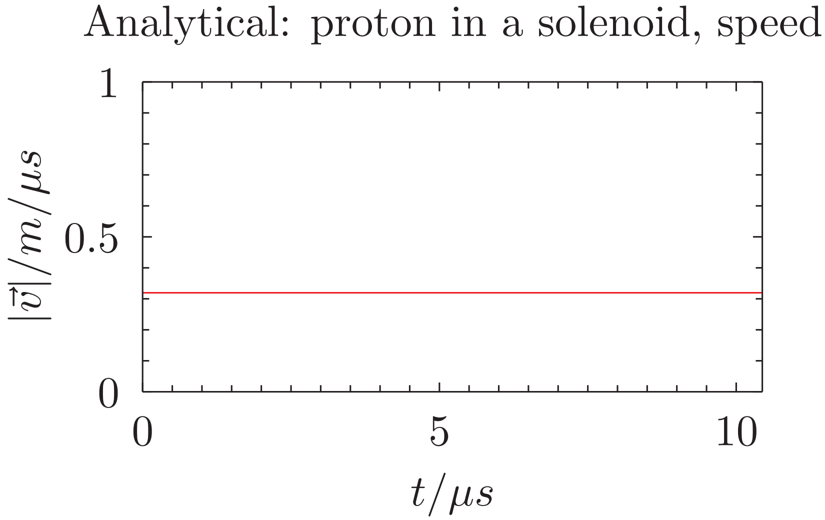
$$R \approx 0.5\,\mathrm{m}\sin(\theta) \qquad T = \frac{2\pi}{\omega_c} \approx 10\,\mu\mathrm{s}$$

Analytical solution: Protons in a Solenoid



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

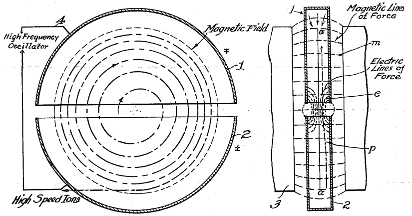
Analytical solution: Protons in a Solenoid



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

Cyclotron accelerator

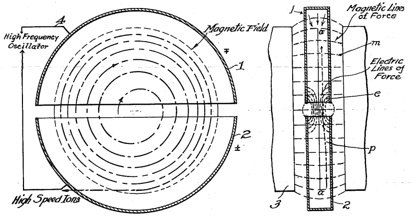
- Electric forces do work.



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

Cyclotron accelerator

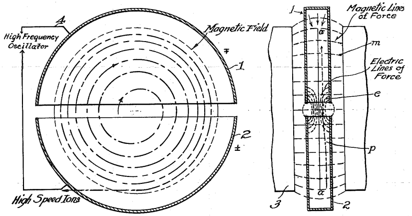
- Electric forces do work.
- Practical example, the Cyclotron.



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.

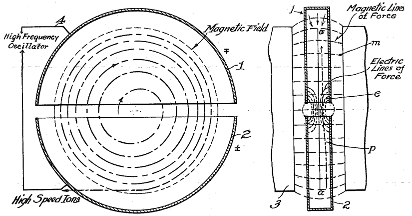


Ernest O. Lawrence, 1934, U.S. Patent 1,948,384; image in Public Domain.

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

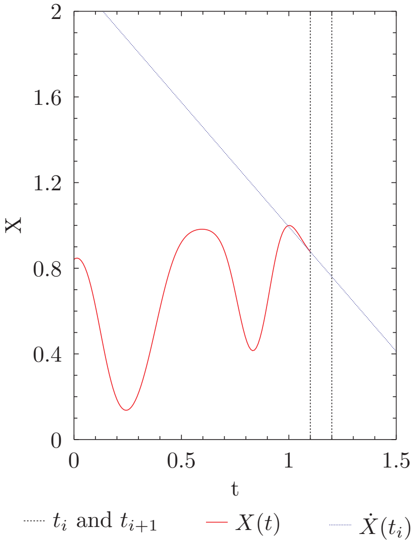
```
auto ODE = [...](const state_type Data, state_type &
    dDatadt, const double t){
     //Extract position and velocity from data
     vec pos = vec(Data[0],Data[1],Data[2]);
     vec velocity = vec(Data[3],Data[4],Data[5]);

     //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.

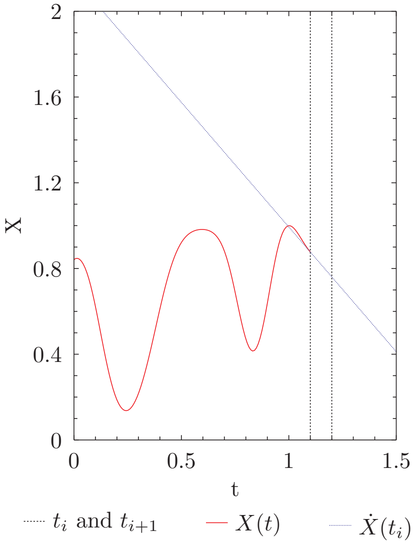


- Bernard P. Zeigler et al. Theory of Modeling and Simulation (Third edition), chapter 3

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.

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



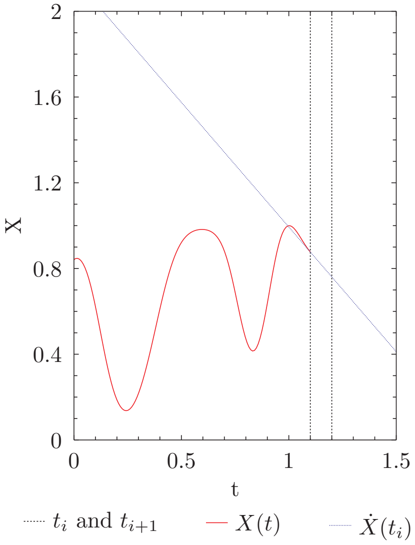
- Bernard P. Zeigler et al. Theory of Modeling and Simulation (Third edition), chapter 3

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 $\mathbf{X}(t_{i+1})$:



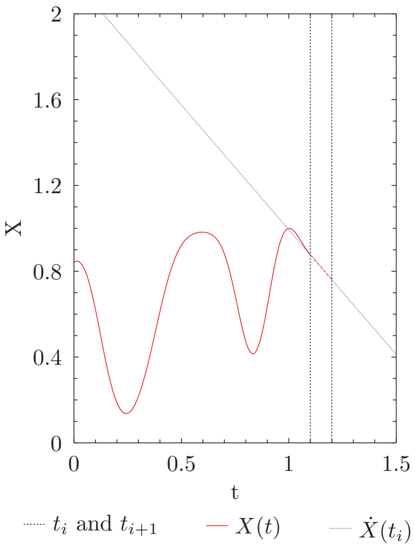
- Bernard P. Zeigler et al. Theory of Modeling and Simulation (Third edition), chapter 3

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

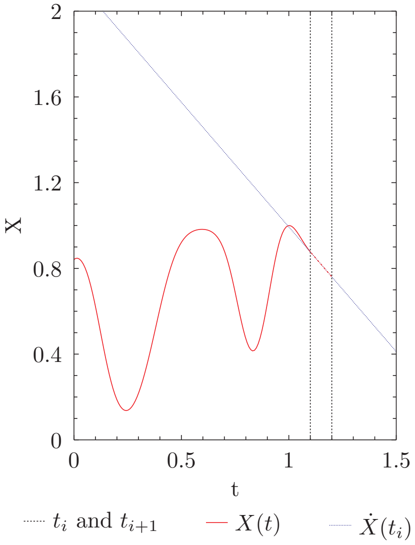
- Let $h = t_{i+1} - t_i > 0$ be constant.
- h, $\mathbf{X}(t)$, t_i and f_{ode} are known.

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

- 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

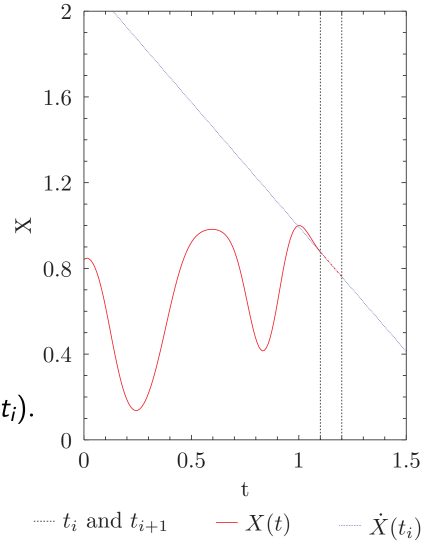
- Let $h = t_{i+1} - t_i > 0$ be constant.
- h, $\mathbf{X}(t)$, t_i and f_{ode} are known.

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

- 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

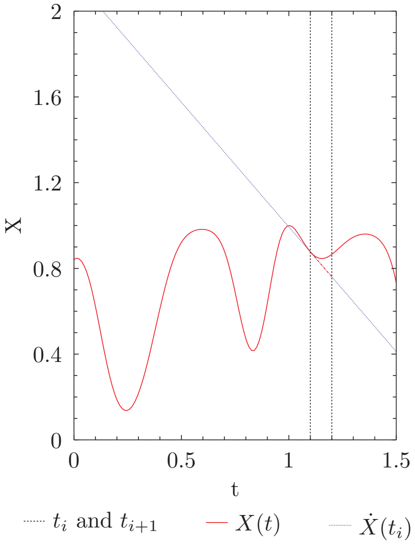
- Let $h = t_{i+1} - t_i > 0$ be constant.
- h, $\mathbf{X}(t)$, t_i and f_{ode} are known.

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

- 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.

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

- *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:

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

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

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

- 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:

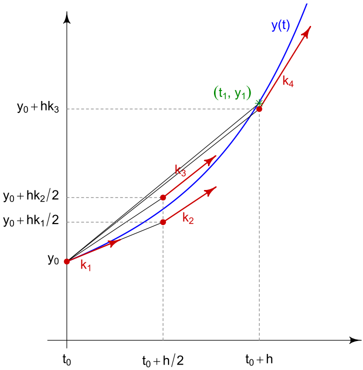
$$\mathbf{X}(t_{i+1}) - \mathbf{X}(t_i) = \frac{h}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)$$

$$\mathbf{k}_1 = f_{ode}(\mathbf{X}(t_i), t_i)$$

$$\mathbf{k}_2 = f_{ode}(\mathbf{X}(t_i) + \frac{h}{2}\mathbf{k}_1, t_i + \frac{h}{2})$$

$$\mathbf{k}_3 = f_{ode}(\mathbf{X}(t_i) + \frac{h}{2}\mathbf{k}_2, t_i + \frac{h}{2})$$

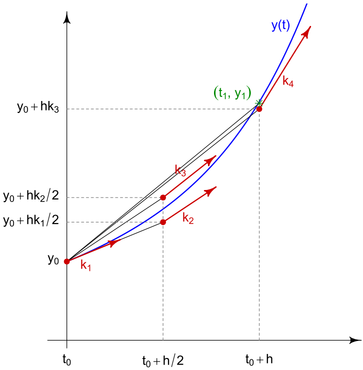
$$\mathbf{k}_4 = f_{ode}(\mathbf{X}(t_i) + h\mathbf{k}_3, t_i + h)$$



The 4th order Runge Kutta method

- RK4, often simply called the Runge Kutta method:

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



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

The General explicit Runge Kutta method

- General explicit, single step, fixed size, Runge Kutta method

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

$$\begin{aligned}
\mathbf{k}_1 &= f_{ode}(\mathbf{X}(t_i), t_i) \\
\mathbf{k}_2 &= f_{ode}(\mathbf{X}(t_i) + h a_{21} \mathbf{k}_1, t_i + c_2 h) \\
\mathbf{k}_3 &= f_{ode}(\mathbf{X}(t_i) + h a_{31} \mathbf{k}_1 + h a_{32} \mathbf{k}_2, t_i + c_3 h) \quad \vdots
\end{aligned}$$

The General explicit Runge Kutta method

- General explicit, single step, fixed size, Runge Kutta method

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

$$\begin{aligned}
\mathbf{k}_1 &= f_{ode}(\mathbf{X}(t_i), t_i) \\
\mathbf{k}_2 &= f_{ode}(\mathbf{X}(t_i) + h a_{21}\mathbf{k}_1, t_i + c_2 h) \\
\mathbf{k}_3 &= f_{ode}(\mathbf{X}(t_i) + h a_{31}\mathbf{k}_1 + h a_{32}\mathbf{k}_2, t_i + c_3 h) \quad \vdots
\end{aligned}$$

- Expressed in Butcher tableu:

$$\begin{array}{c|ccc}
c_1 = 0 & & & \\
c_2 & a_{21} & & \\
c_3 & a_{31} & a_{32} & \\
c_n & a_{n1} & a_{n2} & \dots \\
\hline
 & b_1 & b_2 & \dots
\end{array}$$

Euler Implementations

```
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:

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

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

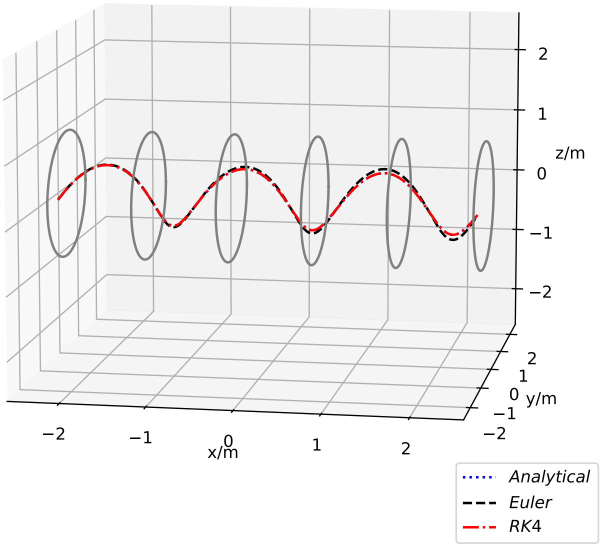
- Test, same proton in a solenoid use $\theta = 60^\circ$ reference, had:

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

- 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

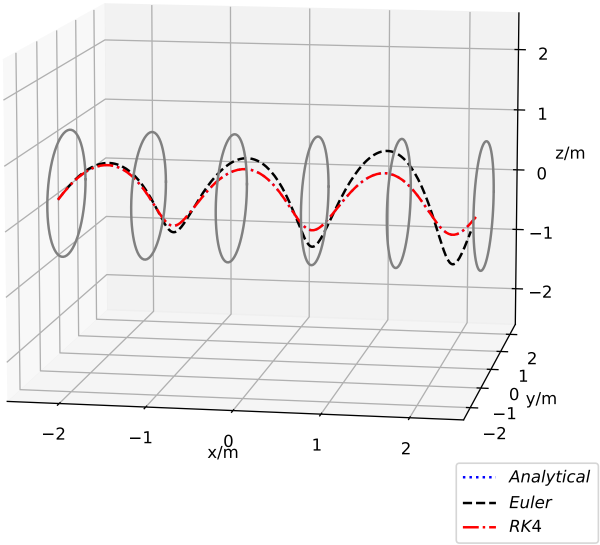
$h = t_{i+1} - t_i = 0.01\,\mu\text{s}$



3129 steps

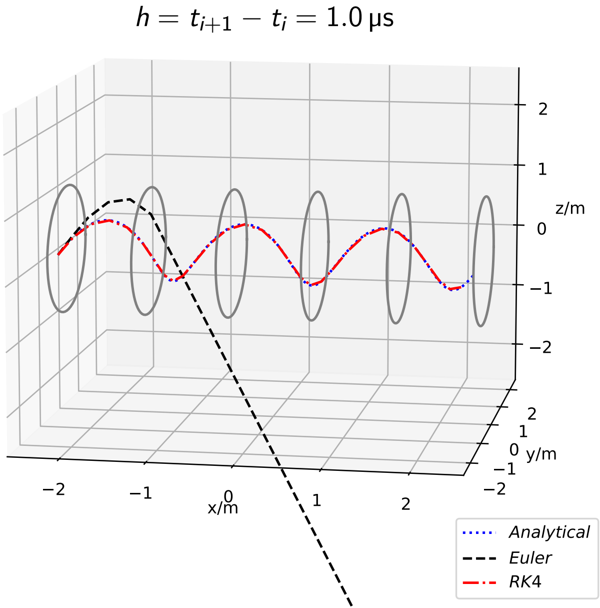
At a glance, 3D view

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



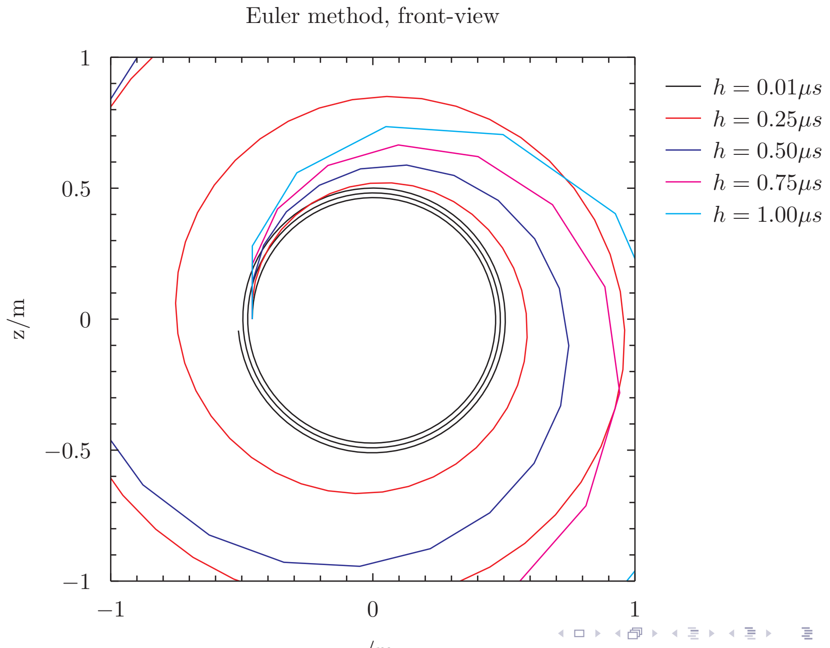
312 steps

At a glance, 3D view

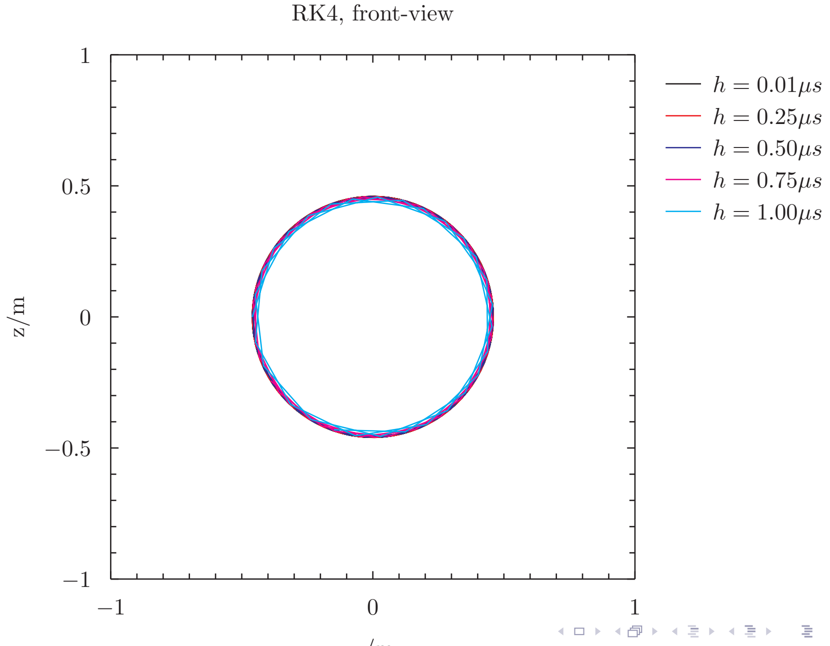


31 steps.

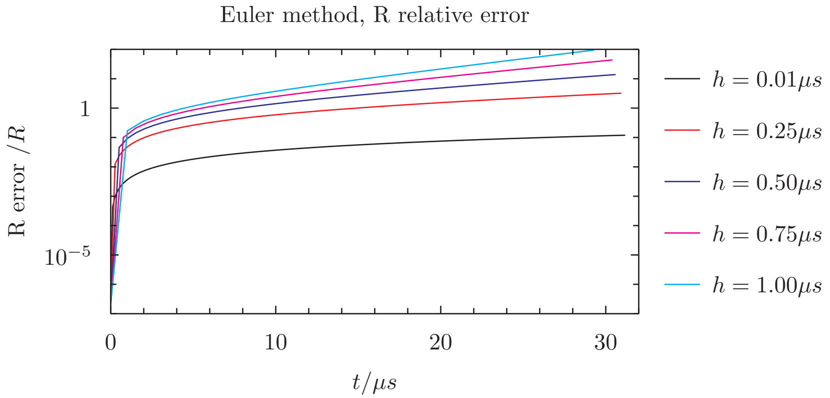
At a glance, front view, no border



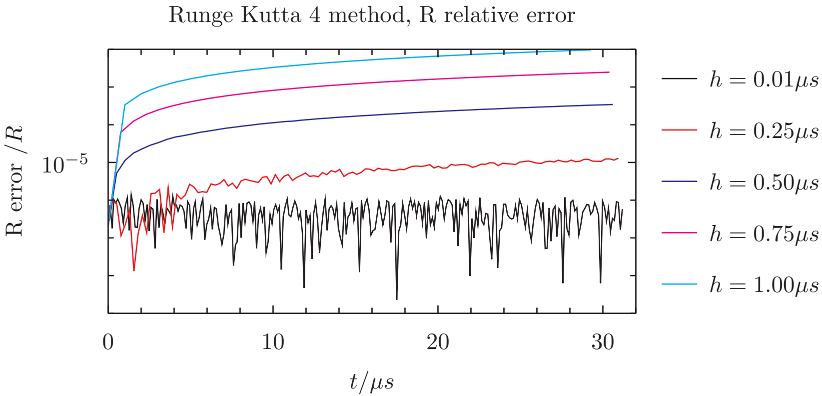
At a glance, front view, no border



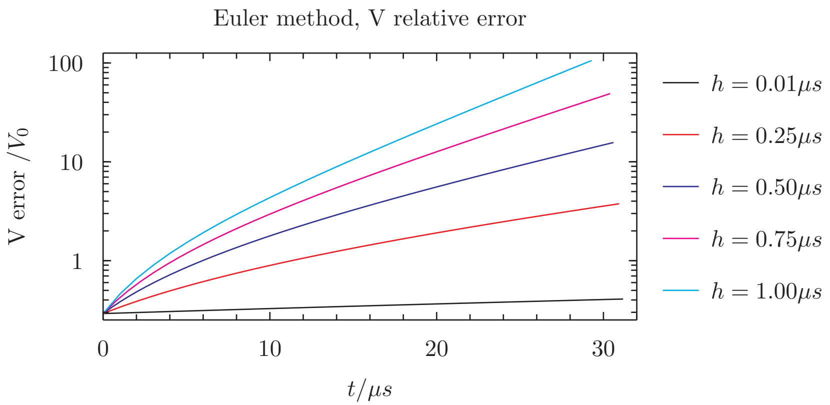
Constant radius?



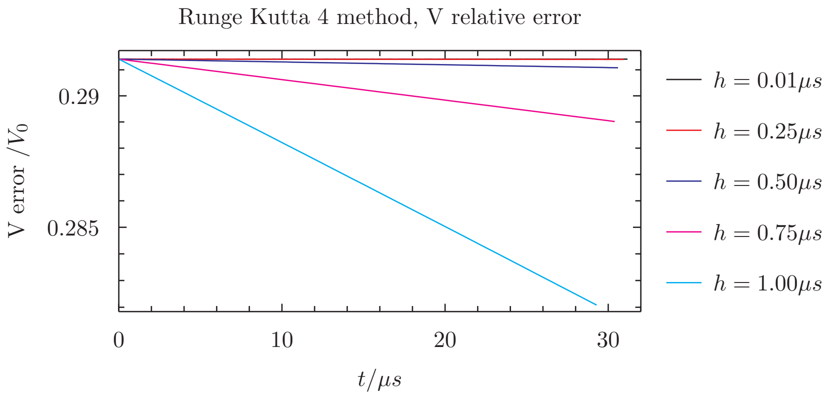
Constant radius?



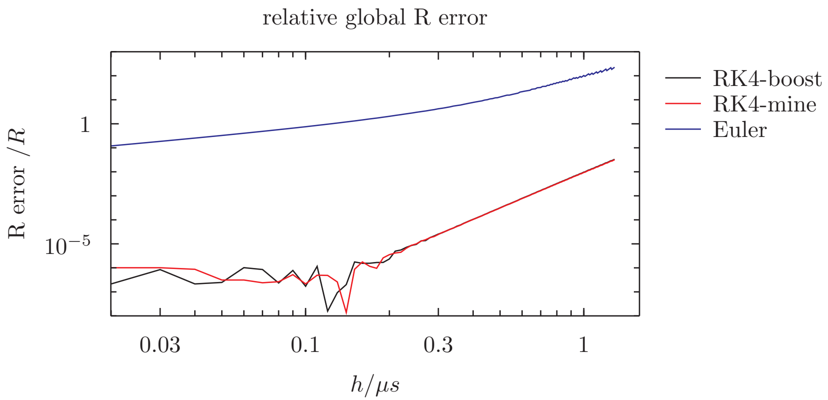
Constant speed?



Constant speed?



Error as function of h



Adaptive step size, why

- h must be small “enough”

Adaptive step size, why

- h must be small “enough”
- Hard to pick, and may change:

Adaptive step size, why

- h must be small "enough"
- Hard to pick, and may change:
- Inhomogeneous fields (here Earth magnetic field)

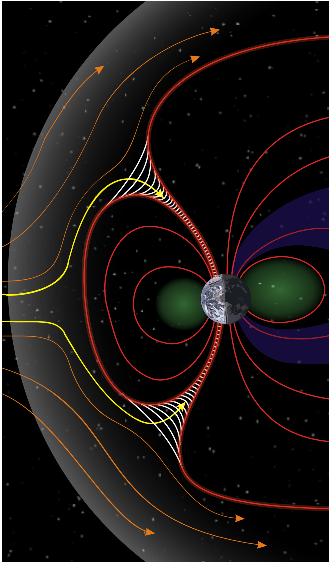
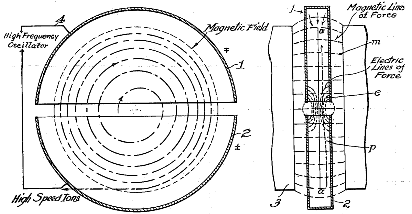


Illustration originally from Nasa. Published on wikipedia, in Public

Adaptive step size, why

- h must be small "enough"
- Hard to pick, and may change:
- Inhomogeneous fields (here Earth magnetic field)
- time dependent fields (here the cyclotron, bad example)



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

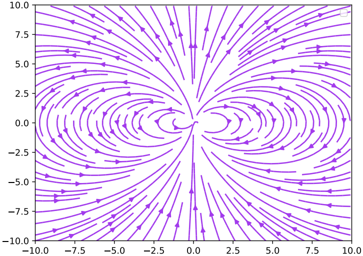
Adaptive step size, why

- h must be small "enough"
- Hard to pick, and may change:
- Inhomogeneous fields (here Earth magnetic field)
- time dependent fields (here the cyclotron, bad example)
- Let the computer pick h.

Example, magnetic dipole

- Not good approximation of the Earth magnetic field.
- True dipole:

$$\vec{B} = \frac{\mu_0}{4\pi r^3}\left[3\hat{\vec{r}}(\vec{m}\cdot\hat{\vec{r}} - \vec{m})\right].$$

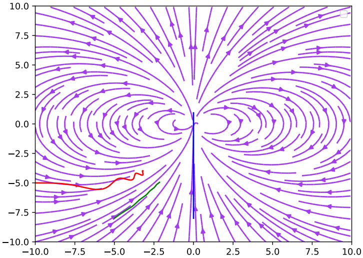


Example, magnetic dipole

- Not good approximation of the Earth magnetic field.
- True dipole:

$$\vec{B} = \frac{\mu_0}{4\pi r^3}\left[3\hat{\vec{r}}(\vec{m}\cdot\hat{\vec{r}} - \vec{m})\right] .$$

- Weak field, little change, long steps.



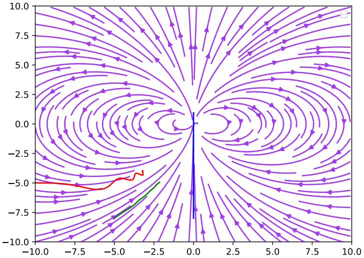
Arbitrarily set $\frac{\mu_0}{4\pi}|\vec{m}| = 0.155\,\mathrm{T}/\mathrm{m}^3$.
Protons with speed around $10\,000\,\mathrm{m}/\mathrm{s}$

Example, magnetic dipole

- Not good approximation of the Earth magnetic field.
- True dipole:

$$\vec{B} = \frac{\mu_0}{4\pi r^3}\left[3\hat{\vec{r}}(\vec{m}\cdot\hat{\vec{r}} - \vec{m})\right] .$$

- Weak field, little change, long steps.
- strong field, large change, short steps.



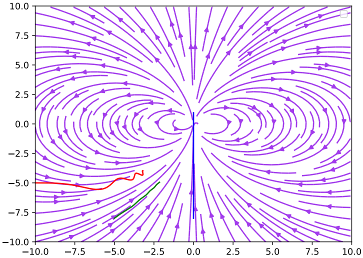
Arbitrarily set $\frac{\mu_0}{4\pi}|\vec{m}| = 0.155\,\mathrm{T/m^3}$.
Protons with speed around $10\,000\,\mathrm{m/s}$

Example, magnetic dipole

- Not good approximation of the Earth magnetic field.
- True dipole:

$$\vec{B} = \frac{\mu_0}{4\pi r^3}\left[3\hat{\vec{r}}(\vec{m}\cdot\hat{\vec{r}} - \vec{m})\right] .$$

- Weak field, little change, long steps.
- strong field, large change, short steps.
- No analytical solution (afaik.)



Arbitrarily set $\frac{\mu_0}{4\pi}|\vec{m}| = 0.155\,\mathrm{T/m^3}$.
Protons with speed around $10\,000\,\mathrm{m/s}$

Runge-Kutta Adaptive step size, how

- Reduce h until the "error" is small enough

Runge-Kutta Adaptive step size, how

- Reduce h until the "error" is small enough
- Use 2 different irder methods, $\mathbf{x}^{(p)}(t_{i+1})$, $\mathbf{x}^{(p)}(t_{i+1})$.

Runge-Kutta Adaptive step size, how

- Reduce h until the "error" is small enough
- Use 2 different irder methods, $\mathbf{x}^{(p)}(t_{i+1})$, $\mathbf{x}^{(p)}(t_{i+1})$.
- Approximate "Error" as $\mathbf{E} = |\mathbf{x}^{(p+1)}(t_{i+1}) - \mathbf{x}^{(p)}(t_{i+1})|$.

Runge-Kutta Adaptive step size, how

- Reduce h until the "error" is small enough
- Use 2 different irder methods, $\mathbf{x}^{(p)}(t_{i+1})$, $\mathbf{x}^{(p)}(t_{i+1})$.
- Approximate "Error" as $\mathbf{E} = |\mathbf{x}^{(p+1)}(t_{i+1}) - \mathbf{x}^{(p)}(t_{i+1})|$.
- Adjust step size to keep the error(s) small (Implementations differ!).

  Dormand, J. R.; Prince, P. J. (1980), "A family of embedded Runge-Kutta formulae", Journal of Computational and Applied Mathematics

(1)

Runge-Kutta Dormand Prince 5 (4)

- `ode45` in Matlab, `scipy.solve_ivp` in Python , `RungeKutta_dopri5` in `boost::odeint`.

$$\mathbf{X}^{(5)}(t_{i+1}) - \mathbf{X}(t_i) = h \sum_{j=1}^{m} b_j^{(5)} \mathbf{k}_j$$

$$\mathbf{X}^{(4)}(t_{i+1}) - \mathbf{X}(t_i) = h \sum_{j=1}^{m} b_j^{(4)} \mathbf{k}_j.$$

$$\mathbf{k}_i = f_{ode}(\mathbf{X}(t_i) + h \sum_{j}^{i-1} a_{ki} \mathbf{k}_i + h a_{32}, t_i + c_3 h).$$

Runge-Kutta Dormand Prince 5 (4)

- `ode45` in Matlab, `scipy.solve_ivp` in Python , `RungeKutta_dopri5` in `boost::odeint`.

$$\mathbf{X}^{(5)}(t_{i+1}) - \mathbf{X}(t_i) = h \sum_{j=1}^{m} b_j^{(5)} \mathbf{k}_j$$

$$\mathbf{X}^{(4)}(t_{i+1}) - \mathbf{X}(t_i) = h \sum_{j=1}^{m} b_j^{(4)} \mathbf{k}_j.$$

$$\mathbf{k}_i = f_{ode}(\mathbf{X}(t_i) + h \sum_{j}^{i-1} a_{ki} \mathbf{k}_i + h a_{32}, t_i + c_3 h).$$

- 7 $\mathbf{k}_i$'s (actually 6).

  Dormand, J. R.; Prince, P. J. (1980), "A family of embedded Runge-Kutta formulae", Journal of Computational and Applied Mathematics

Butcher Tableu of Dormand Prince (5) 4

$$
\begin{array}{c|cccccccc}
c_i & a_{ij} & \dots & & & & & & \\
\hline
0 & & & & & & & & \\
\frac{1}{5} & \frac{1}{5} & & & & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & & & \\
\frac{4}{5} & \frac{40}{45} & -\frac{56}{15} & -\frac{32}{9} & & & & & \\
\frac{8}{9} & \frac{19372}{6561} & -\frac{25360}{2187} & \frac{64448}{6561} & -\frac{212}{729} & & & & \\
1 & \frac{9017}{3168} & -\frac{355}{33} & \frac{46732}{5247} & \frac{49}{176} & -\frac{5103}{18656} & & & \\
1 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & & \\
\hline
/\,b^{(5)} & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & 0 & \\
/\,b^{(4)} & \frac{5179}{57600} & 0 & \frac{7571}{16695} & & \frac{393}{640} & -\frac{92097}{339200} & \frac{187}{2100} & \frac{1}{40}
\end{array}
$$

Runge-Kutta Dormand Prince 5 (4)

- Step size correction (Dormand Prince):

$$h_{new} = 0.9 h_{old} \left[\frac{\delta}{||\mathrm{E}||} \right]^{\frac{1}{p+1}}$$

Runge-Kutta Dormand Prince 5 (4)

- Step size correction (Dormand Prince):

$$h_{new} = 0.9 h_{old} \left[\frac{\delta}{||\mathrm{E}||} \right]^{\frac{1}{p+1}}$$

- Step size correction (Me):

$$h_{new} = \min_j \left(0.9 h_{old} \left[\frac{\delta_j}{\mathrm{E}_j} \right]^{\frac{1}{p}} \right), \quad \mathrm{E}_j > \delta_j : \mathrm{reject}$$

$$\delta_j = \min(\delta_{abs}, |\mathbf{X}_j(t_i)|\delta_{rel}) \sqrt{\frac{h_{old}}{T}}.$$

Runge-Kutta Dormand Prince 5 (4)

- Step size correction (Dormand Prince):

$$h_{new} = 0.9 h_{old} \left[\frac{\delta}{||\mathrm{E}||} \right]^{\frac{1}{p+1}}$$

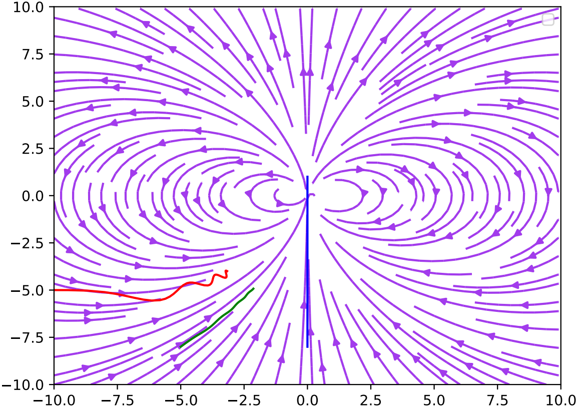
- Step size correction (Me):

$$h_{new} = \min_j \left(0.9 h_{old} \left[\frac{\delta_j}{\mathrm{E}_j} \right]^{\frac{1}{p}} \right), \quad \mathrm{E}_j > \delta_j : \mathrm{reject}$$

$$\delta_j = \min(\delta_{abs}, |\mathbf{X}_j(t_i)|\delta_{rel}) \sqrt{\frac{h_{old}}{T}}.$$

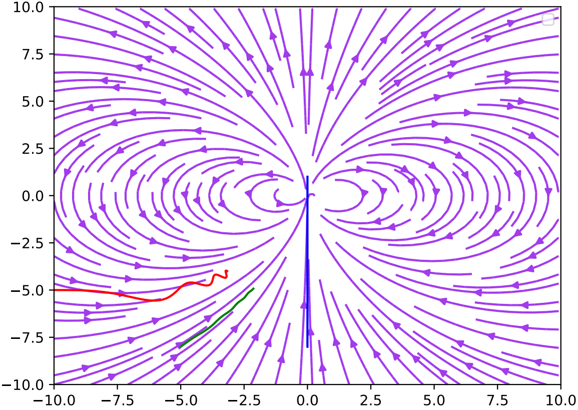
- Relative and absolute error at end. "Fail safe" $\sqrt{\ldots}$.
  Dormand, J. R.; Prince, P. J. (1980), "A family of embedded Runge-Kutta formulae", Journal of Computational and Applied Mathematics

Does it work?



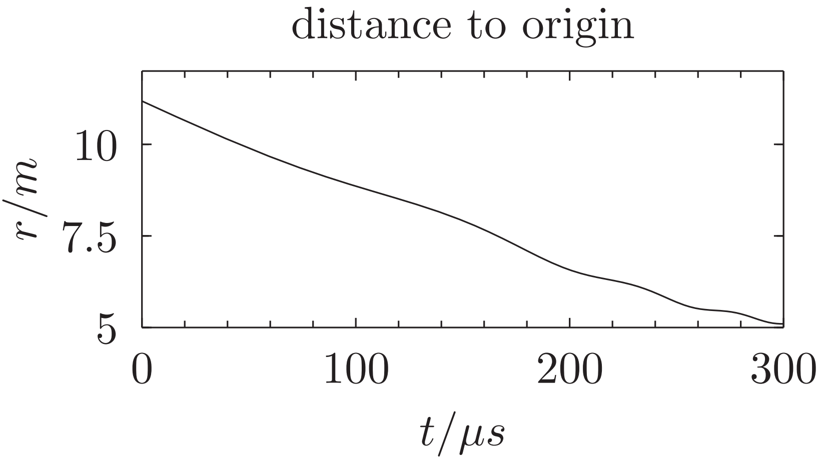
My version relative and absolute error 10^{-6}. 94 steps (+ 14 rejected).

Does it work?

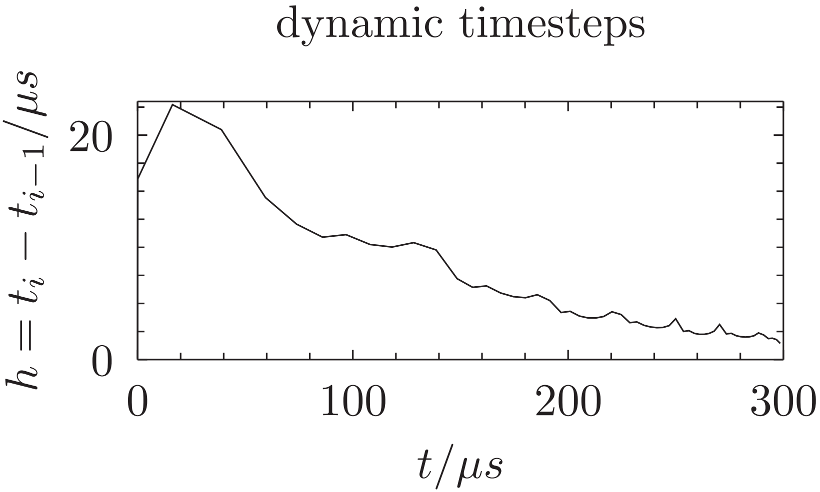


Odeint library relative and absolute error 10^{-7}. 92 steps.

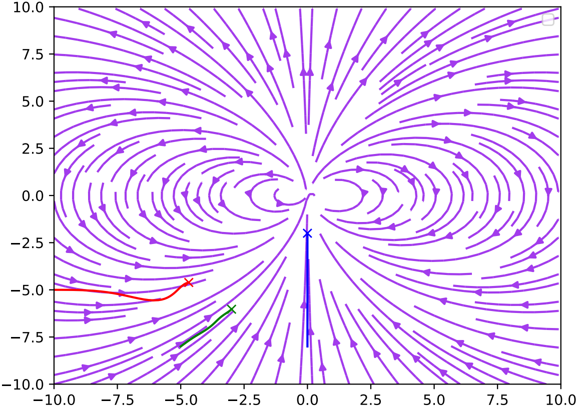
Does it work?



Does it work?

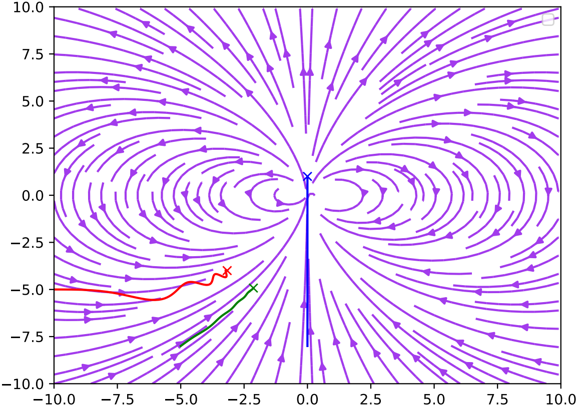


Another curious result



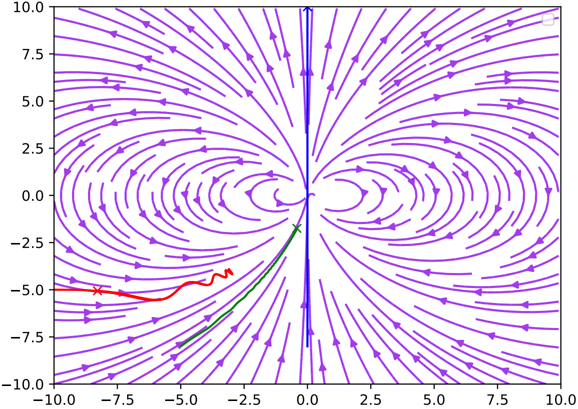
$T = 0.2\,\mathrm{ms}$

Another curious result



$T = 0.3\,\mathrm{ms}$

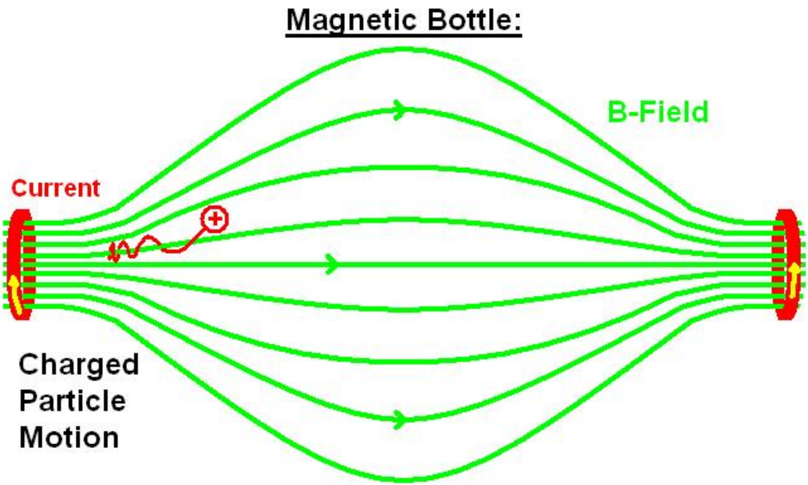
Another curious result



$$T = 0.6\,\text{ms}$$

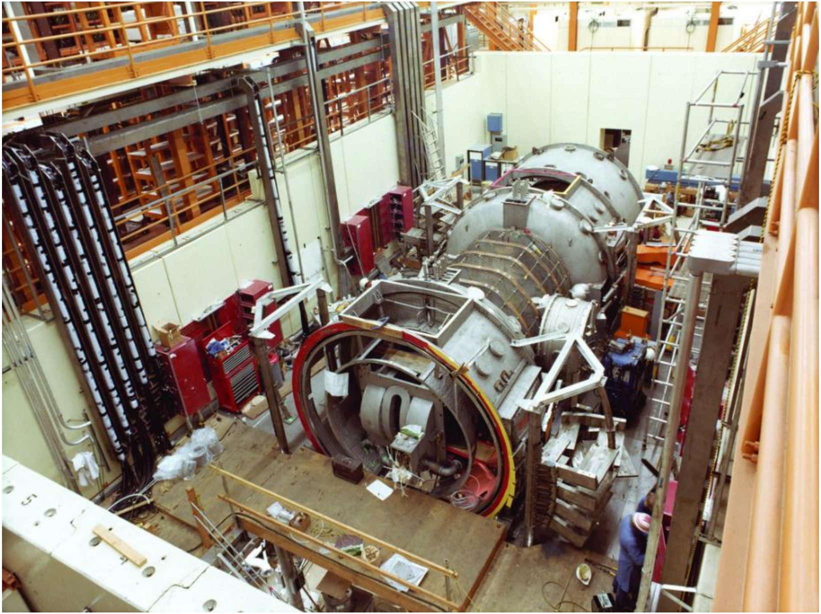
Magnetic “mirror” or “bottle”.

Another curious result



Wikipedia user WikiHelper2134 , Public Domain.

Another curious result



Tandem Mirror Experiment, The Lawrence Livermore National

Extra examples-"if time permits"

- The cyclotron
  - When the Dormand Prince method fails

Extra examples-“if time permits”

- The cyclotron
  - When the Dormand Prince method fails
- Toroidal fields
  - Another non-analytic setup.

Conclusion

- Simulating particles in electric and magnetic fields.

Conclusion

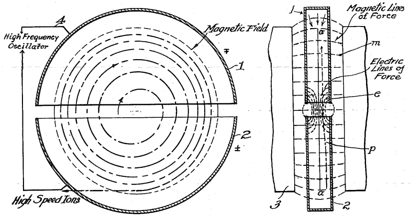
- Simulating particles in electric and magnetic fields.
- Numerically solving ordinary differential equations
- Reasonable agreement with known results.
- Can easily be generalized to other systems.

Conclusion

- Simulating particles in electric and magnetic fields.
- Numerically solving ordinary differential equations
- Reasonable agreement with known results.
- Can easily be generalized to other systems.
- Limitation, simulations are not experiments.

Example, cyclotron

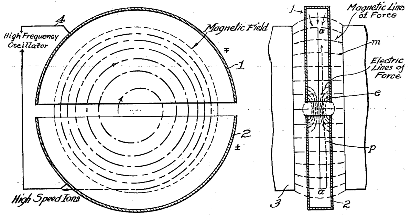
- Electric field accelerates, magnetic contains.



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

Example, cyclotron

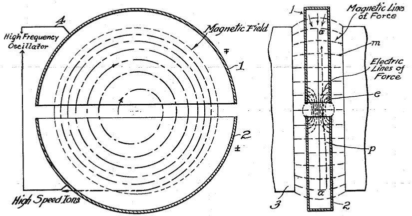
- Electric field accelerates, magnetic contains.
- Single gab, oscillating field.



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

Example, cyclotron

- Electric field accelerates, magnetic contains.
- Single gab, oscillating field.
- Uses classical Cyclotron frequency

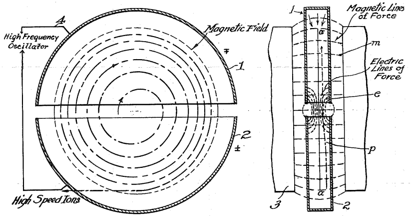


Ernest O. Lawrence, 1934, U.S. Patent 1,948,384; image in Public Domain.

Example, cyclotron

- Electric field accelerates, magnetic contains.
- Single gab, oscillating field.
- Uses classical Cyclotron frequency
- Analytical final speed, in principle path.

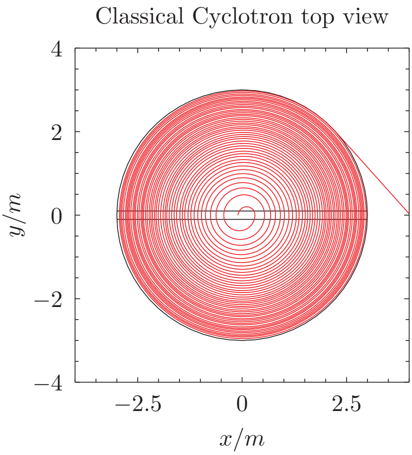
$$\frac{R|q|B}{m} = v_\perp$$



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

Can it be simulated

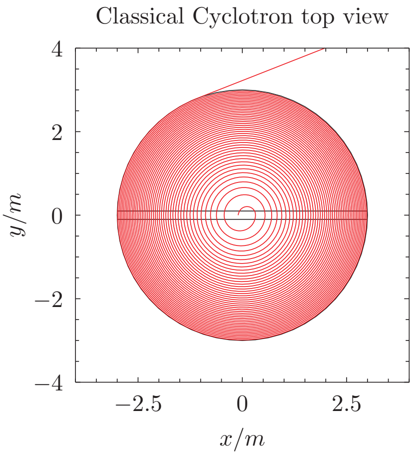
- With fixed step size, looks bad 4999 points



4999 points.

Can it be simulated

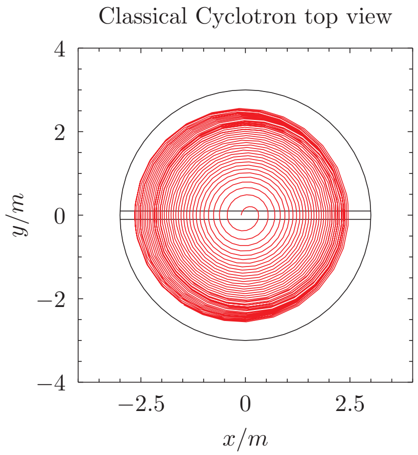
- With fixed step size, looks bad 4999 points
- My adabtive method, error: 10^{-6}



500000 points.

Can it be simulated

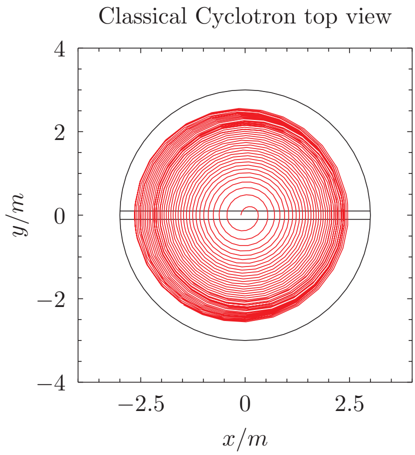
- With fixed step size, looks bad 4999 points
- My adabtive method, error: 10^{-6}
- Odeint library



2070 points.

Can it be simulated

- With fixed step size, looks bad 4999 points
- My adabtive method, error: 10^{-6}
- Odeint library
- Non-continuous ODE are bad.



2070 points.