How physics simulations work: simulating particles in electric and magnetic fields

Nikolaj Roager Christensen

Student Colloquium in Physics and Astronomy, Aarhus University

March 2021

Numerical integration of ordinary differential equations: motion of charged particles in electromagnetic fields

Introduction

Theory and physical background 10 min

Euler's Method and the 4th order Runge-Kutta Method 15 min
 Euler's Method
 Higher order Runge-Kutta methods
 Demonstration, particles in a solenoid

Embedded algorithms, and adaptive step size 15 min
 Dormand Prince 5 (4) method
 Demonstration: magnetic dipole

Conclusion

- Numerical simulations are important.

- ▶ Numerical simulations are important.
- ▶ Testing setups, non-analytical systems.

- Numerical simulations are important.
- Testing setups, non-analytical systems.
- Demonstration, charged particles in electric and magnetic fields.
- Analytically known and not.

- Numerical simulations are important.
- Testing setups, non-analytical systems.
- Demonstration, charged particles in electric and magnetic fields.
- Analytically known and not.
- Simulations are not experiments!

- Some repetition from Electrodynamics

Theory: Classical non-relativistic particles

- ▶ Some repetition from Electrodynamics
- ▶ The Lorentz force+N2:

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

Theory: Classical non-relativistic particles

▶ Some repetition from Electrodynamics
▶ The Lorentz force+N2:

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

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

Theory: Classical non-relativistic particles

▶ Some repetition from Electrodynamics
▶ The Lorentz force+N2:

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

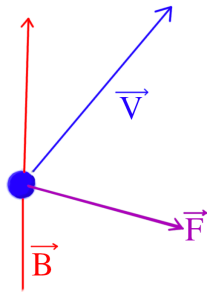▶ Only 1 particle! so pre-programmed depending on the setup.
▶ Could use potentials $\phi(\vec{r}, t)$ $\vec{A}(\vec{r}, t)$ and Hamiltonian, or Lagrangian.
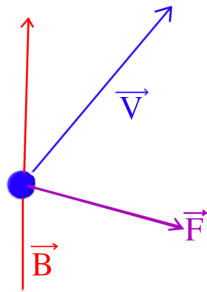▶ Other systems would have other differential equations.

▶ Magnetic forces do no work:

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

► Magnetic forces do no work:

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

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

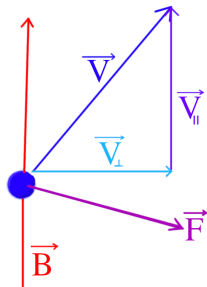$$|\vec{F}_B| = |q(\vec{v} \times \vec{B})| = |qv_\perp B|.$$
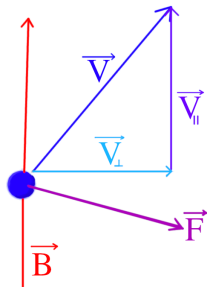
Known results, cyclotron motion $\vec{B}$ fields

▶ 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

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

▶ Magnetic forces do no work:
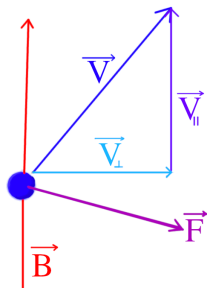
$$dW_{\vec{B}} = \vec{F}_B \cdot d\vec{r} \propto (\vec{v} \times \vec{B}) \cdot \vec{v} = 0.$$
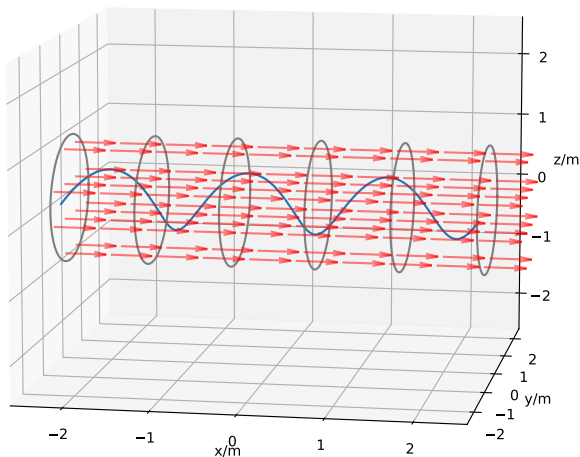
▶ $(\vec{v} = \vec{v}_\perp + \vec{v}_\parallel)$:

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

▶ Same as Centripetal force:
Cyclotron motion

▶ Cyclotron radius and
frequency:

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

What we expect to see

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

What we expect to see
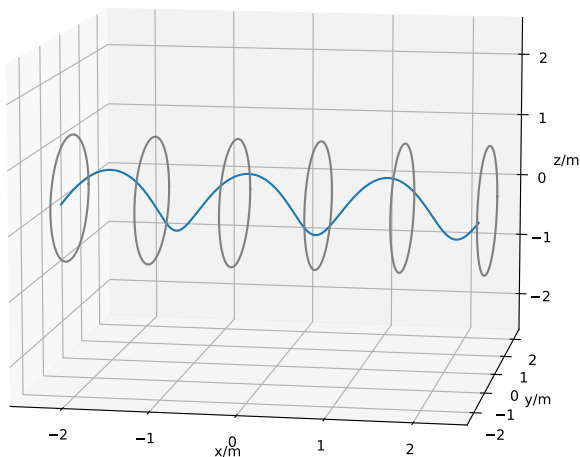
"Cyclotron motion"

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

What we expect to see

Analytical: proton in a solenoid, side/front-view

Analytical: proton in a solenoid, speed

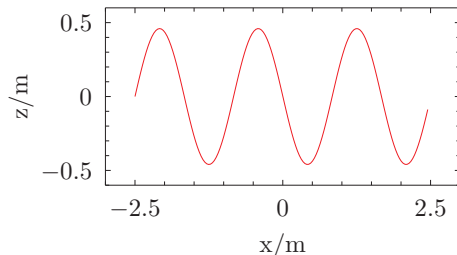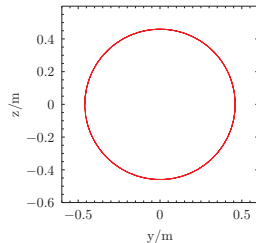"Cyclotron motion"

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

What we expect to see
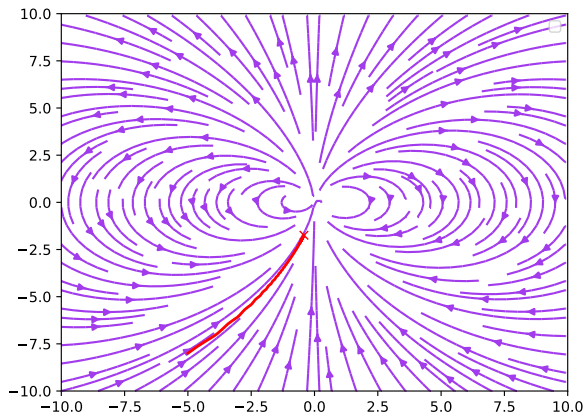
(Actually from my simulation)

Ordinary differential equation's.

▶ Sources: Zeigler et al. Theory of Modeling and Simulation (Third edition) chapter 3

▶ We have:

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

Ordinary differential equation's.

- Sources: Zeigler et al. Theory of Modeling and Simulation (Third edition) chapter 3
- We have:

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

- 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

▶ We have:

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

▶ Algorithms exists for ODEs:

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

▶ Here:

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

The ODE to solve

```cpp
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 V = vec(Data[3],Data[4],Data[5]);
    //Lorentz+Newtons 2nd law
    vec F = Charge*(Fields.get_Efield(pos,t)+
        cross(V,Fields.get_Bfield(pos,t)));
    vec dVdt = F*Inv_mass;
    //Save derivative of data
    dDatadt[0]=V.x;dDatadt[1]=V.y;dDatadt[2]=V.z;
    dDatadt[3]=dVdt.x;dDatadt[4]=dVdt.y;dDatadt[5]=
    dVdt.z;
    }
```

Ordinary differential equation's.

- Sources: Zeigler et al. Theory of Modeling and Simulation (Third edition) chapter 3
- We have:

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

- Algorithms exists for ODEs:

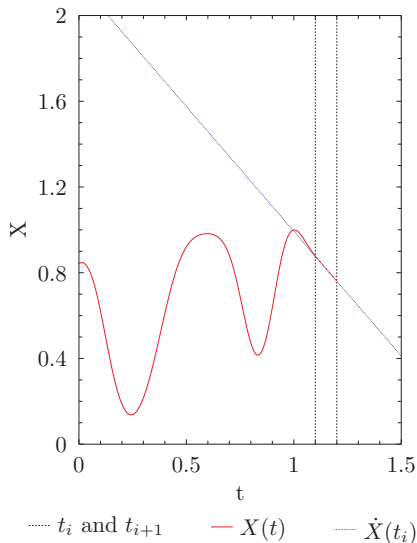$$\dot{\mathbf{X}} = f_{ode}(\mathbf{X}(t), t).$$

- Here:

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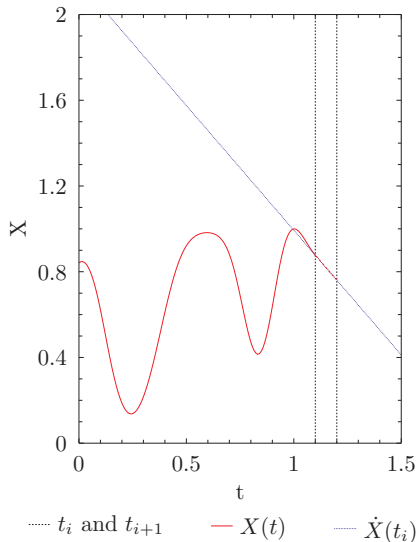
Solving differential equations

▶ We know only $\mathbf{X}(t_0)$ and t_0 and f_{ode}.

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

Solving differential equations

- We know only $\mathbf{X}(t_0)$ and t_0 and f_{ode}.
- Can we find $\mathbf{X}(t_0 + h)$ for $h > 0$?.

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

$\cdots\cdots$ t_i and t_{i+1} $\quad$ — $X(t)$ $\quad$ $\cdots\cdots$ $\dot{X}(t_i)$

Solving differential equations

- We know only $\mathbf{X}(t_0)$ and t_0 and f_{ode}.
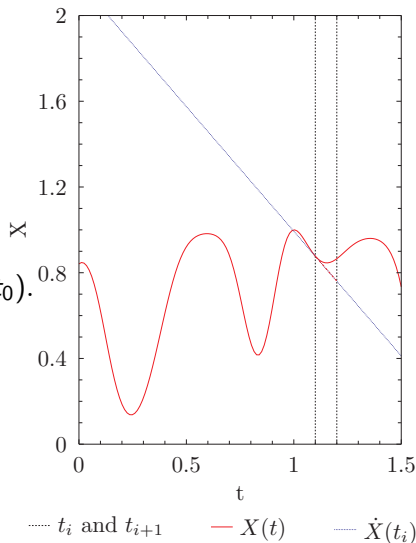
- Can we find $\mathbf{X}(t_0 + h)$ for $h > 0$?.

- (Explicit Forward) Euler's Method:

$$\mathbf{X}(t_0+h) = \mathbf{X}(t_0)+hf_{ode}(\mathbf{X}(t_0), t_0).$$

- Multiple steps $t_0, t_1 = t_0 + h, \ldots t_i, \ldots$.

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

$\cdots\cdots$ t_i and t_{i+1} — $X(t)$ $\cdots\cdots$ $\dot{X}(t_i)$

Solving differential equations

- We know only $\mathbf{X}(t_0)$ and t_0 and f_{ode}.
- Can we find $\mathbf{X}(t_0 + h)$ for $h > 0$?.
- (Explicit Forward) Euler's Method:

$$\mathbf{X}(t_0+h) = \mathbf{X}(t_0)+hf_{ode}(\mathbf{X}(t_0), t_0).$$

- Multiple steps $t_0, t_1 = t_0 + h, \ldots t_i, \ldots$.
- Can this be justified?
- Bernard P. Zeigler et al. Theory of Modeling and Simulation (Third edition), chapter 3
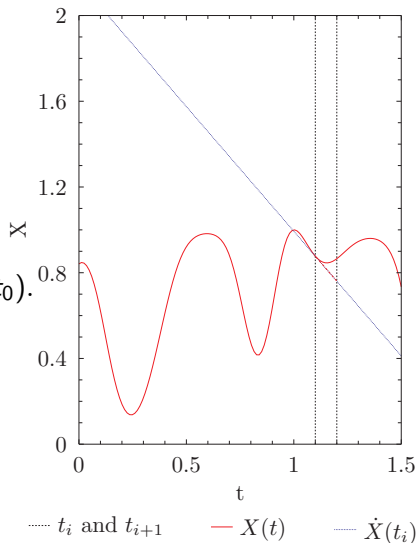
$\cdots\cdots\; t_i \text{ and } t_{i+1}$ $\rule{1cm}{0.4pt}\; X(t)$ $\cdots\cdots\; \dot{X}(t_i)$

▶ Multiple justifications for why.

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

$$\mathbf{X}(t_0 + h) = \mathbf{X}(t_0) + hf_{ode}(\mathbf{X}(t_0), t_0) + h^2 \ldots + \ldots.$$

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

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

- Local error h^2. Global error: h^1.

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

$$\mathbf{X}(t_i + h) = \mathbf{X}(t_i) + h f_{ode}(\mathbf{X}(t_i), t_i) + h^2 \ldots + \ldots$$

- Local error h^2. Global error: h^1.
- Argument suggests we need $\dot{f}_{ode}, \ddot{f}_{ode}, \ldots$

▶ In general:

$$\mathbf{X}(t_{i+1}) - \mathbf{X}(t_i) = \int_{t_i}^{t_{i+1}} \dot{\mathbf{X}}(t_i) dt.$$

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

The Runge Kutta steppers

- In general:

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

- *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:

The Runge Kutta steppers

▶ 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:

The Runge Kutta steppers

- In general:

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

- *Mean Value theorem for integrals* $t_i \leq \tau \leq t_{i+1}$.
- Guess $\tau = t_i$. Good guess, if $\dot{f}_{ode} \approx 0$.

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

The Runge Kutta steppers

- In general:

$$\mathbf{X}(t_{i+1}) - \mathbf{X}(t_i) = hf_{ode}(\mathbf{X}(\tau), \tau).$$

- *Mean Value theorem for integrals* $t_i \leq \tau \leq t_{i+1}$.
- Guess $\tau = t_i$. Good guess, if $\dot{f}_{ode} \approx 0$.
- More generally, (*Explicit* and *single step*), Runge-Kutta family:

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

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

The Runge Kutta steppers

- In general:

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

- *Mean Value theorem for integrals $t_i \leq \tau \leq t_{i+1}$.*
- Guess $\tau = t_i$. Good guess, if $\dot{f}_{ode} \approx 0$.
- More generally, (*Explicit* and *single step*), Runge-Kutta family:

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

$$= \sum_{j=1}^{m} h_j f_{ode}(\mathbf{X}(\tau_j), \tau_j).$$

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

Runge Kutta methods

- More commonly written:

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

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

Runge Kutta methods

- More commonly written:

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

- Estimate $\mathbf{K}_j$:

$$\mathbf{K}_1 = f_{ode}(\mathbf{X}(t_i), t_i)$$
$$\mathbf{K}_2 = f_{ode}(\mathbf{X}(t_i) + h a_{21} \mathbf{K}_1, t_i + c_2 h)$$
$$\mathbf{K}_3 = f_{ode}(\mathbf{X}(t_i) + h a_{31} \mathbf{K}_1 + h a_{32} \mathbf{K}_2, t_i + c_3 h)$$
$$\vdots$$

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

Runge Kutta methods

- More commonly written:

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

- Estimate $\mathbf{K}_j$:

$$\mathbf{K}_1 = f_{ode}(\mathbf{X}(t_i), t_i)$$
$$\mathbf{K}_2 = f_{ode}(\mathbf{X}(t_i) + ha_{21}\mathbf{K}_1, t_i + c_2 h)$$
$$\mathbf{K}_3 = f_{ode}(\mathbf{X}(t_i) + ha_{31}\mathbf{K}_1 + ha_{32}\mathbf{K}_2, t_i + c_3 h)$$
$$\vdots$$

- Want exact for p'th order polynomial.

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

Runge Kutta methods

- More commonly written:

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

- Estimate $\mathbf{K}_j$:

$$\mathbf{K}_1 = f_{ode}(\mathbf{X}(t_i), t_i)$$
$$\mathbf{K}_2 = f_{ode}(\mathbf{X}(t_i) + h a_{21}\mathbf{K}_1, t_i + c_2 h)$$
$$\mathbf{K}_3 = f_{ode}(\mathbf{X}(t_i) + h a_{31}\mathbf{K}_1 + h a_{32}\mathbf{K}_2, t_i + c_3 h)$$
$$\vdots$$

- Want exact for p'th order polynomial.
- Taylor series analogy: error $\ h^p$.
- Martha L. Abell, James P. Braselton, Differential Equations with Mathematica (Fourth Edition), 2016:

- 2nd order (Heun's method):

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

Higher order methods

► 4th order, often simply called the Runge Kutta method:

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

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

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

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

$$\mathbf{k}_4 = f_{ode}(\mathbf{X}(t_i) + h\mathbf{k}_3, t_i + h)$$
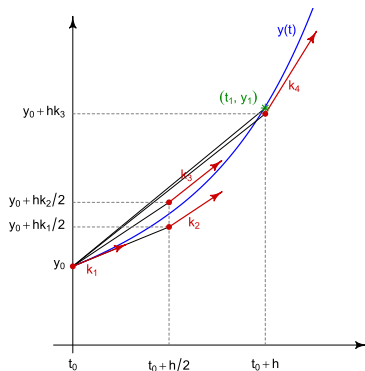
▶ General explicit, single step, fixed size, Runge Kutta method

- General explicit, single step, fixed size, Runge Kutta method
- Expressed in Butcher tableu:

$c_1 = 0$			
c_2	a_{21}		
c_3	a_{31}	a_{32}	
c_n	a_{n1}	a_{n2}	$\ldots$
	b_1	b_2	$\ldots$

Euler Implementations

```cpp
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;

    save_step( Data , i*timestep );
};
```

```
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);

    //substep 2
    temp=Data+timestep*K1/2;
    ODE(temp,K2,t+timestep/2);
```

```
    //substep 3
    temp=Data+timestep*K2/2;
    ODE(temp,K3,t+timestep/2);

    //substep 4
    temp=Data+timestep*K3;
    ODE(temp,K4,t+timestep);

    //Read data
    Data+=timestep*(K1+2.0*K2+2.0*K3+K4)/6.0;
    save_step( Data , i*timestep );
}
```

"Correct" way

```cpp
#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
);
```

▶ Test, same proton in a solenoid use $\theta = 60°$ reference, had:

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

Does it work

▶ Test, same proton in a solenoid use $\theta = 60°$ reference, had:

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

▶ Compare Analytic, Euler, Runge-Kutta 4.

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

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

- Compare Analytic, Euler, Runge-Kutta 4.
- Consider $\theta = 60°$ at different h.

Does it work

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

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

- Compare Analytic, Euler, Runge-Kutta 4.
- Consider $\theta = 60°$ at different h.
- Check error on $|\vec{v}|$, $R = \sqrt{y^2 + z^2}$ and $x(t)$.
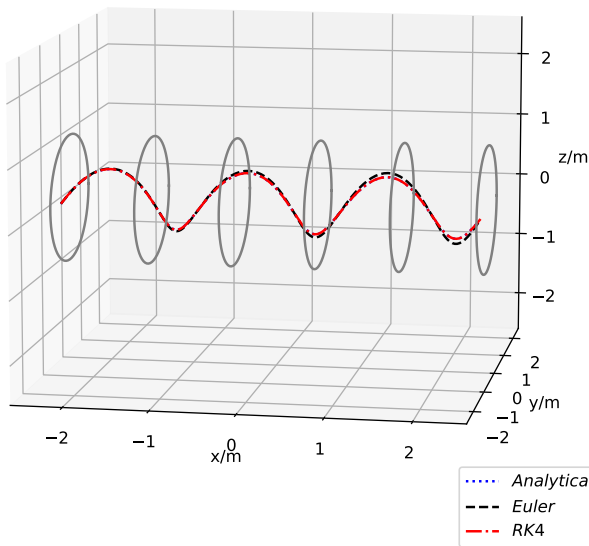
$h = t_{i+1} - t_i = 0.01 \, \mu s$

3129 steps
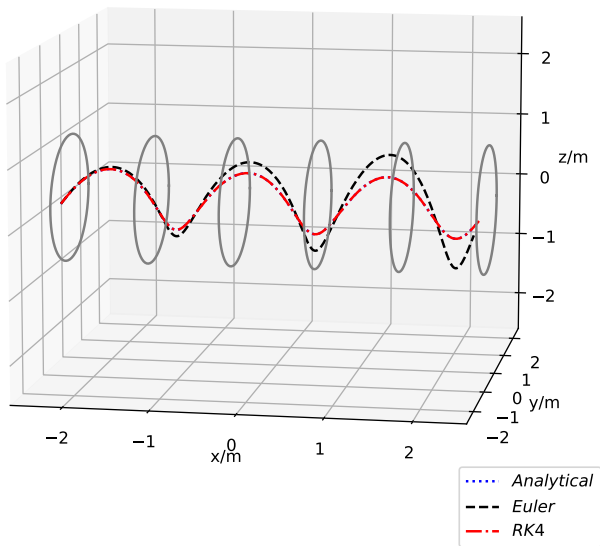
$$h = t_{i+1} - t_i = 0.1\,\mu\text{s}$$

312 steps

$h = t_{i+1} - t_i = 1.0\,\mu s$
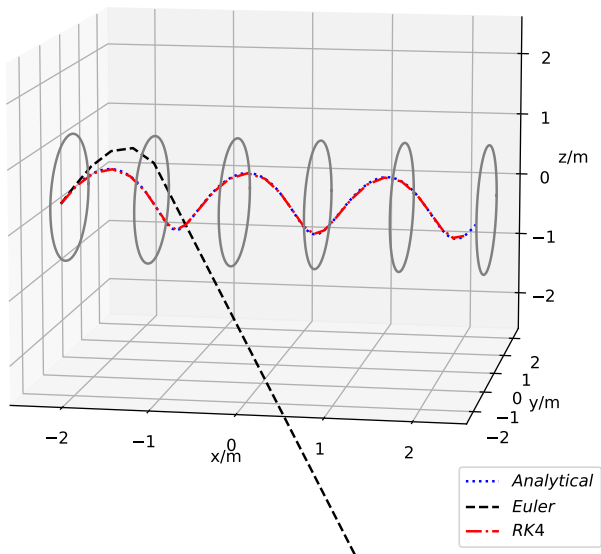
31 steps.

Euler method, front-view
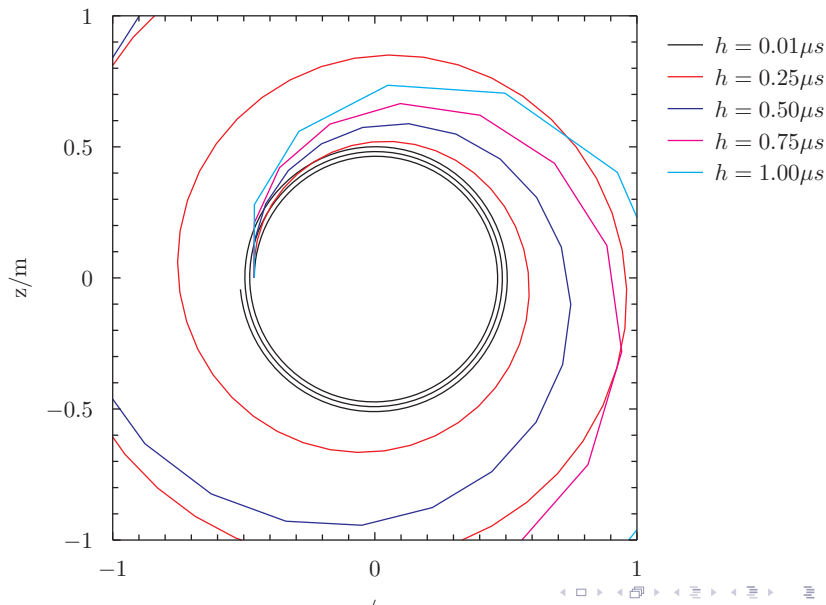
RK4, front-view
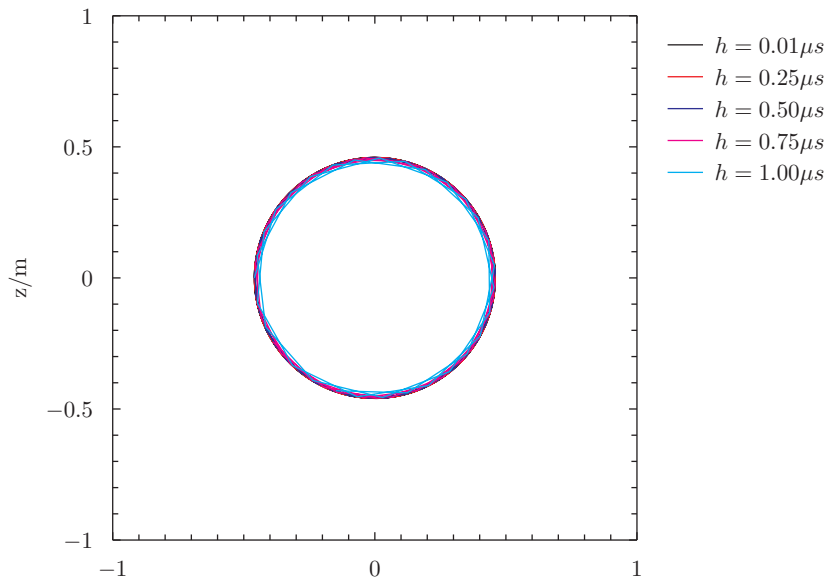
$h = 0.01 \mu s$
$h = 0.25 \mu s$
$h = 0.50 \mu s$
$h = 0.75 \mu s$
$h = 1.00 \mu s$

Constant radius?

Euler method, R relative error

Runge Kutta 4 method, R relative error

relative global R error

- h must be small "enough"

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

- h must be small "enough"
- Hard to pick, and may change:
- Inhomogeneous fields (here, a true dipole)
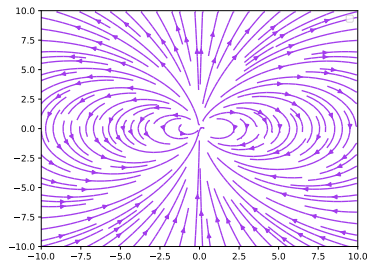
- h must be small "enough"
- Hard to pick, and may change:
- Inhomogeneous fields (here, a true dipole)
- Time dependent fields (here the cyclotron, bad example)

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

- h must be small "enough"
- Hard to pick, and may change:
- Inhomogeneous fields (here, a true dipole)
- Time dependent fields (here the cyclotron, bad example)
- Let the computer pick h.
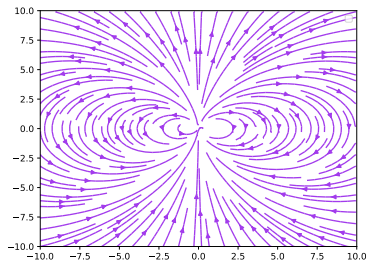
Example, magnetic dipole

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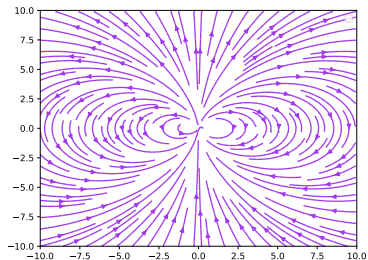
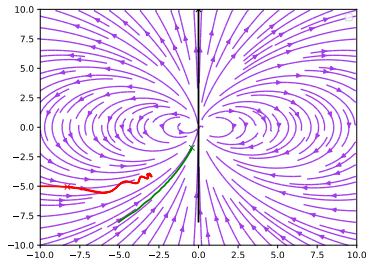$$\vec{B} = \frac{\mu_0}{4\pi r^3} \left[3\hat{r}(\vec{m} \cdot \hat{r}) - \vec{m} \right].$$

Example, magnetic dipole

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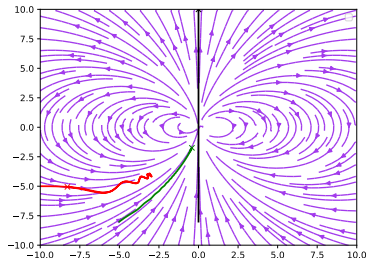$\frac{\mu_0}{4\pi}|\vec{m}| = 0.155\,\text{T/m}^3$.
Protons with speed around
$10\,000\,\text{m/s}$

Example, magnetic dipole

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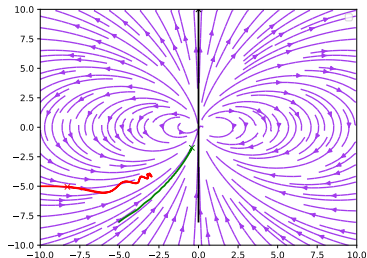$\frac{\mu_0}{4\pi}|\vec{m}| = 0.155\,\text{T/m}^3$.
Protons with speed around
$10\,000\,\text{m/s}$

Example, magnetic dipole

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

- ▶ Reduce h until the "error" is small enough

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

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

$$(1)$$

Runge-Kutta Dormand Prince 5 (4)

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

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

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

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

Runge-Kutta Dormand Prince 5 (4)

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

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

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

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

- 7 $\mathbf{k}_i$'s (actually 6 by clever re-usage).
  Dormand, J. R.; Prince, P. J. (1980), "A family of embedded Runge-Kutta formulae", Journal of Computational and Applied Mathematics

c_i	a_{ij}	$\dots$					
0							
$\frac{1}{5}$	$\frac{1}{5}$						
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$					
$\frac{4}{5}$	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$				
$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$			
1	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$		
1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	
$/\,b^{(5)}$	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0
$/\,b^{(4)}$	$\frac{5179}{57600}$	0	$\frac{7571}{16695}$	$\frac{393}{640}$	$-\frac{92097}{339200}$	$\frac{187}{2100}$	$\frac{1}{40}$

▶ 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):

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

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

Runge-Kutta Dormand Prince 5 (4)

- Step size correction (Dormand Prince):

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

- Step size correction (Me):

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

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

- Scale by step size $\frac{h_{old}}{T}$, "Fail safe" $\sqrt{\dots}$.

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

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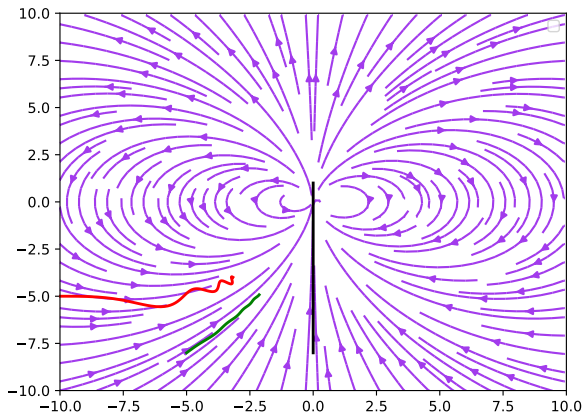
My version relative and absolute error 10^{-6}. 94 steps ($+$ 14 rejected).

Does it work?
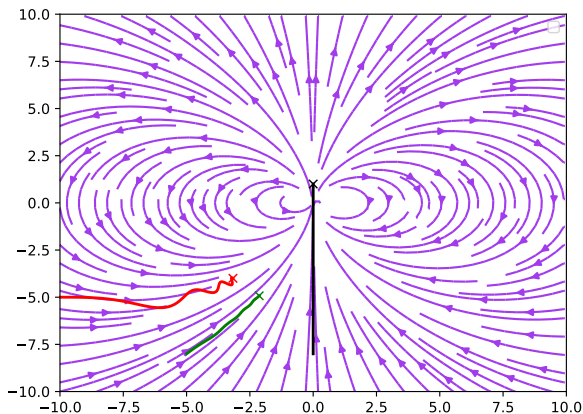
Odeint library relative and absolute error 10^{-7}. 92 steps.
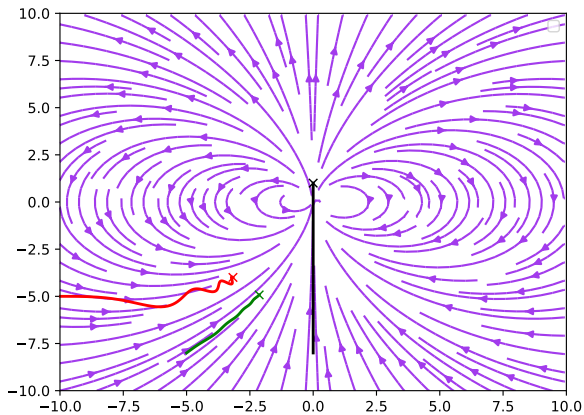
distance to origin
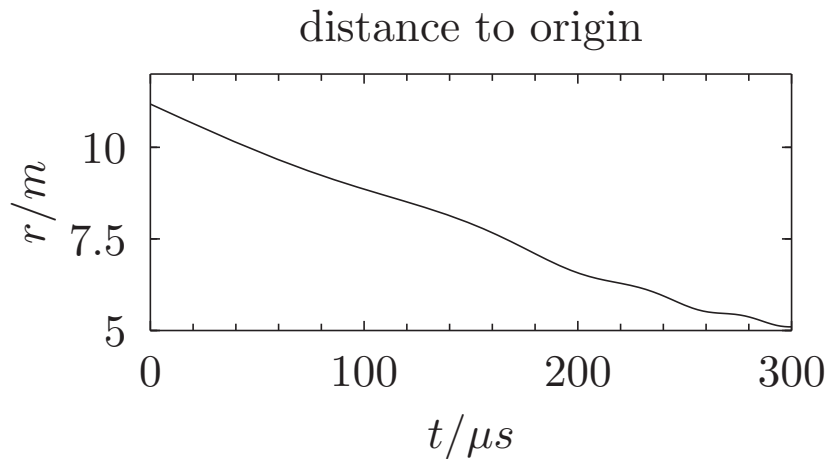
Adaptive timesteps
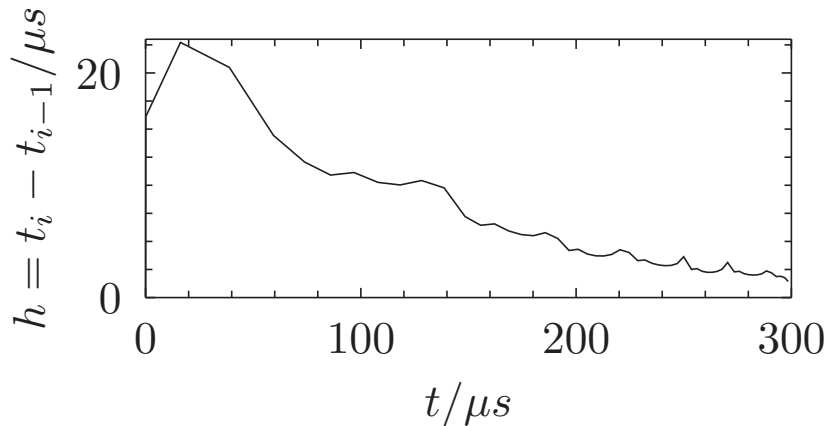
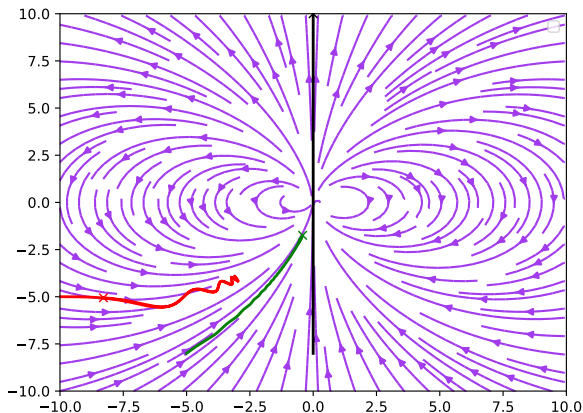

My version, adaptive step size.

Another curious result
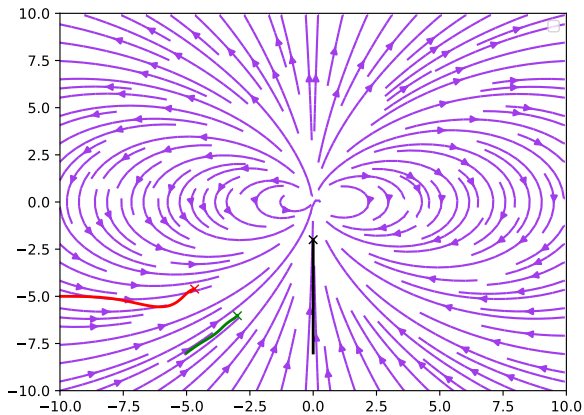
$T = 0.2\,\text{ms}$

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

Another curious result

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

Magnetic "mirror" or "bottle".

Magnetic Bottle:
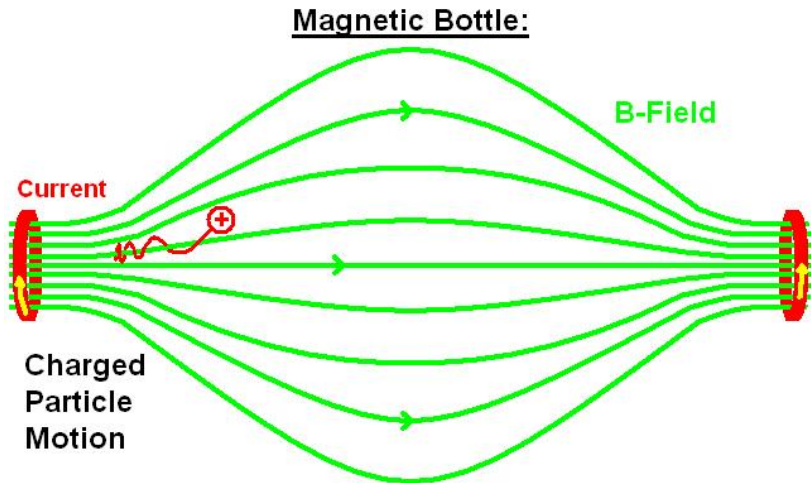
Wikipedia user WikiHelper2134 , Public Domain.
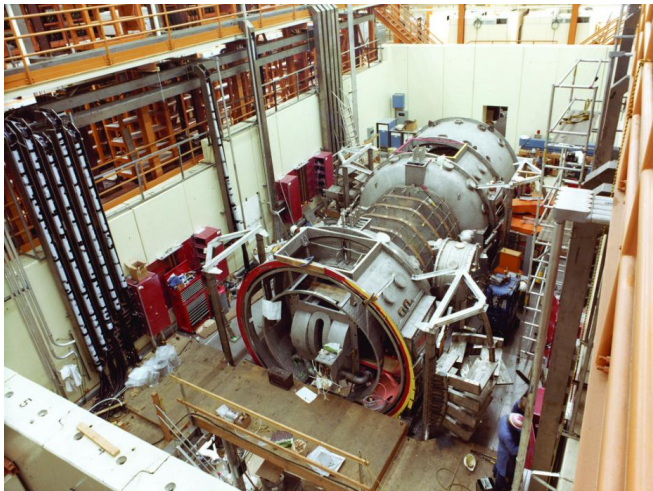
Tandem Mirror Experiment, The Lawrence Livermore National
Laboratory, 1978.

▶ 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.
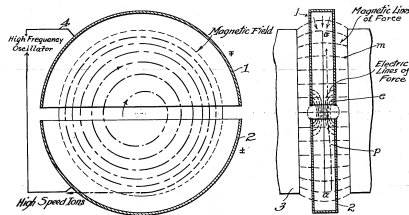- Limitation, simulations are not experiments.

- Electric field accelerates, magnetic contains.
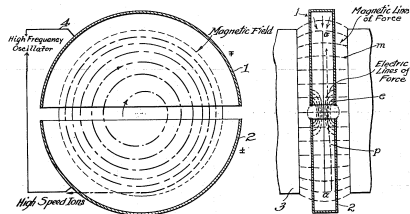
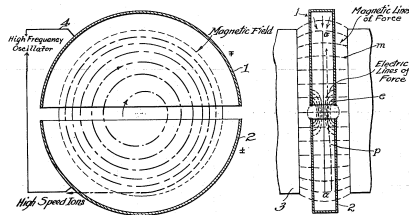Ernest O. Lawrence, 1934, U.S. Patent 1,948,384; image in Public Domain.

Example, cyclotron

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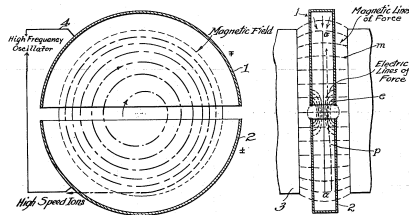

Ernest O. Lawrence, 1934, U.S. Patent 1,948,384; image in Public Domain.

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
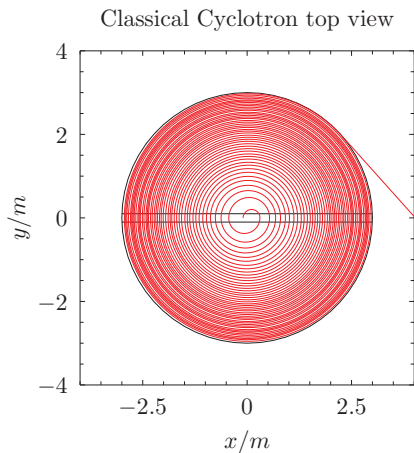- Analytical final speed, in principle path.

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

Ernest O. Lawrence, 1934, U.S. Patent 1,948,384; image in Public Domain.
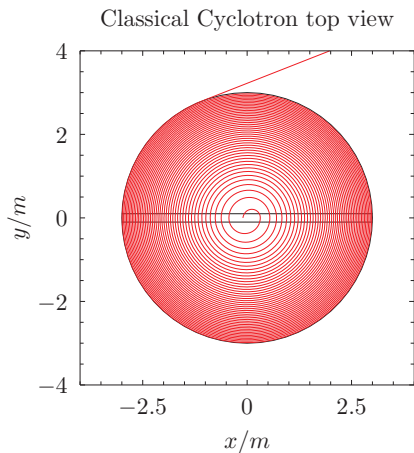
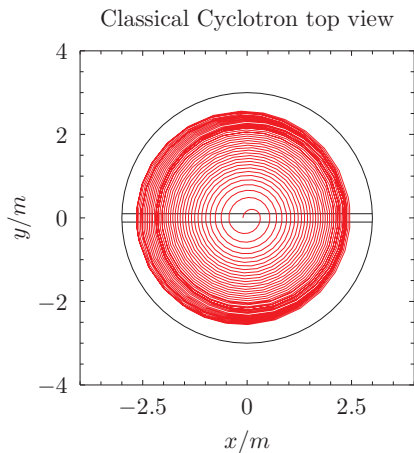- with fixed step size, looks bad 4999 points

Classical Cyclotron top view

can it be simulated

- with fixed step size, looks bad 4999 points
- my adabtive method, error: 10^{-6}
- Looks great but 2 million points.

Classical Cyclotron top view

- with fixed step size, looks bad 4999 points
- my adabtive method, error: 10^{-6}
- Looks great but 2 million points.
- odeint library

Classical Cyclotron top view

- with fixed step size, looks bad 4999 points
- my adabtive method, error: 10^{-6}
- Looks great but 2 million points.
- odeint library
- non-continuous ode are bad.

Classical Cyclotron top view