COMPUTATIONAL PHYSICS

Numerical methods

Ordinary Differential Equations

Partial Differential Equations

COMPUTATIONAL PHYSICS

Numerical methods

Ordinary Differential Equations

- ✓ *Euler method*
- ✓ *Runge-Kutta method*
- ✓ *Adaptive methods*
- ✓ *Boundary value problems*

Ordinary Differential Equations

- In ordinary differential equations, only derivatives with respect to a single variable are involved e.g.

$$\frac{dx}{dt} = g(t,x) \quad \longrightarrow \quad \frac{dN}{dt} = -\alpha N$$

Exponential radioactive decay law

- Unsurprisingly, higher order equations can be cast as a system of differential equations of first order…

$$\frac{d^2x}{dt^2} = g(t,x,\frac{dx}{dt}) \quad \longrightarrow \quad \frac{d^2x}{dt^2} + \frac{\lambda}{m}\frac{dx}{dt} + \frac{k}{m}x = 0$$

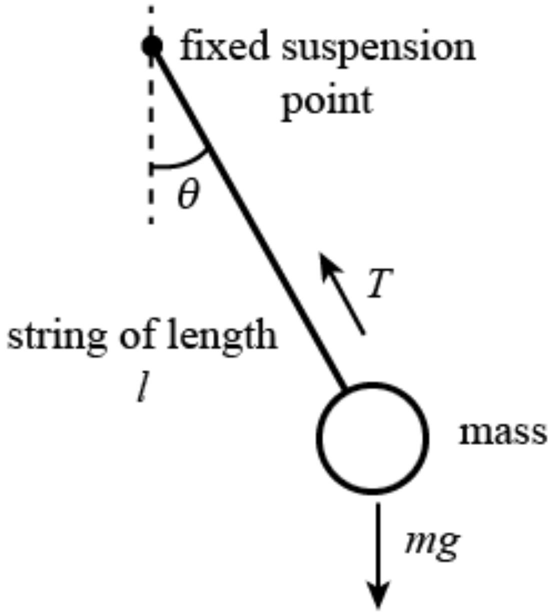
Damped harmonic oscillator

…is the same as

$$\begin{cases} \dfrac{dx}{dt} = y \\ \dfrac{dy}{dt} = g(t,x,y) \end{cases} \Leftrightarrow \begin{cases} \dfrac{dy^{(0)}}{dt} = f^{(1)}(t,y^{(0)},y^{(1)}) \\ \dfrac{dy^{(1)}}{dt} = f^{(2)}(t,y^{(0)},y^{(1)}) \end{cases} \Leftrightarrow \begin{bmatrix} \dfrac{dy^{(0)}}{dt} \\ \dfrac{dy^{(1)}}{dt} \end{bmatrix} = \begin{bmatrix} f^{(1)}(t,y^{(0)},y^{(1)}) \\ f^{(2)}(t,y^{(0)},y^{(1)}) \end{bmatrix}$$

Notable Examples

Good old pendulum…



$$\frac{d^2\theta}{dt^2} + \frac{g}{l}\sin(\theta) = 0$$

Elasticity over a bar…

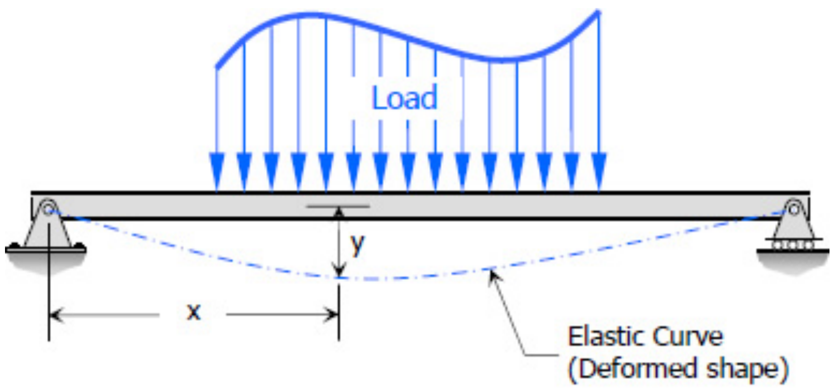


Figure: Elastic curve

https://www.mathalino.com

$$\frac{d^2}{dx^2}\left(EI\frac{d^2y}{dx^2}\right) = w(x)$$

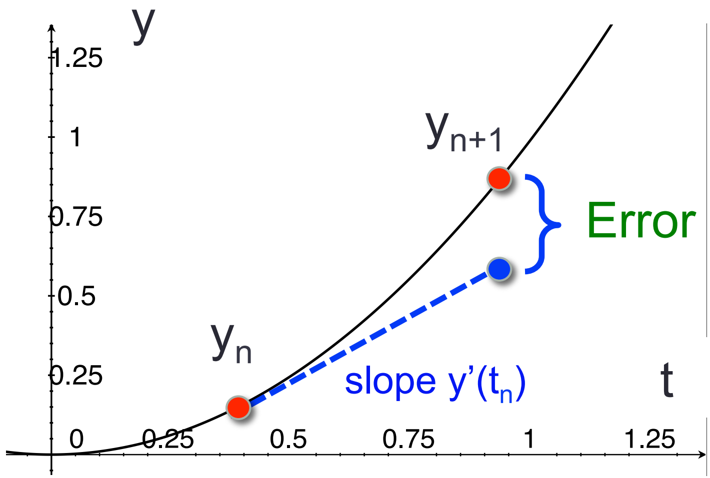
1^{st} order ODE: Euler method

- ODE and "formal solution"

$$\boxed{\frac{d\vec{y}}{dt} = \vec{f}(t,\vec{y}) \Leftrightarrow \vec{y}(t) = \vec{y}(t_0) + \int_{t_0}^{t} \vec{f}(t,\vec{y})\,dt}$$

- **Euler method (1st order accurate)**

$$\frac{\vec{y}_{n+1} - \vec{y}_n}{\delta t} \cong \frac{d\vec{y}}{dt} = \vec{f}(t,\vec{y}) \Leftrightarrow \boxed{\vec{y}_{n+1} = \vec{y}_n + \delta t \vec{f}(t_n,\vec{y}_n) + \mathrm{O}((\delta t)^2)}$$



...what about stability ? (simpler 1D case)

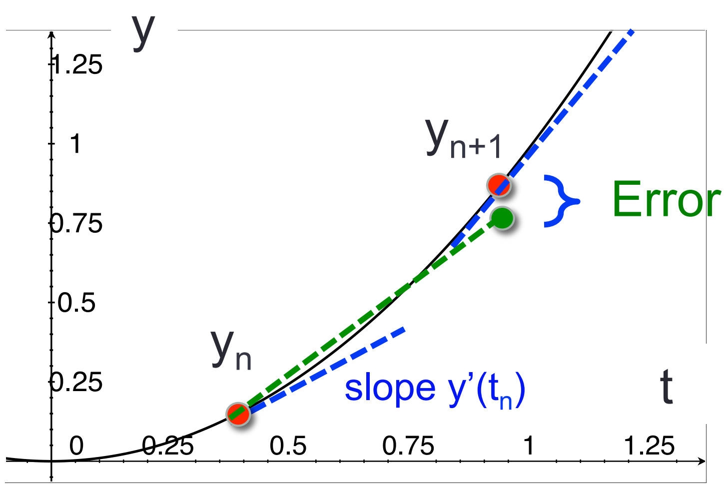
→ Consider y_n is already containated with error and see how it propagates to new iteration !

$$\cancel{y_{n+1}} + \delta y_{n+1} = \cancel{y_n} + \delta y_n + \delta t(\cancel{f(t_n, y_n)} + \left.\frac{\partial f}{\partial y}\right|_n \delta y_n) \boxed{\Leftrightarrow \delta y_{n+1} = \delta y_n \left(1 + \delta t \left.\frac{\partial f}{\partial y}\right|_n\right)} \rightarrow \boxed{\left|1 + \delta t \left.\frac{\partial f}{\partial y}\right|_n\right| < 1}$$

1st order ODE: Trapezoidal method (CN)

- Integrate using trapezoidal rule

$$\boxed{\vec{y}_{n+1} = \vec{y}_n + \int_{t_n}^{t_{n+1}} \vec{f}(t,\vec{y})dt = \vec{y}_n + \frac{\delta t}{2}\left(\vec{f}(t_n,\vec{y}_n) + \vec{f}(t_{n+1},\vec{y}_{n+1})\right)}$$



Average the slope !!!

- **Implicit method (2nd order accurate)**

$$\boxed{\vec{y}_{n+1} = \vec{y}_n + \frac{\delta t}{2}\left(\vec{f}(t_n,\vec{y}_n) + \vec{f}(t_{n+1},\vec{y}_{n+1})\right) + O((\delta t)^3)}$$

...how can i get y_{n+1} needing y_{n+1} ?!

→ **IMPLICIT** methods

→ If f(t,y) is non-linear...one might as well approximate it

$$\boxed{y^*_{n+1} = y_n + \delta t f(t_n, y_n)}$$

$$\boxed{\vec{y}_{n+1} = \vec{y}_n + \frac{\delta t}{2}\left(\vec{f}(t_n,\vec{y}_n) + \vec{f}(t_{n+1},\vec{y}^*_{n+1})\right) + O((\delta t)^3)}$$

Heun method

→ Stable if

$$\boxed{\left|\frac{1+\delta t \left.\frac{\partial f}{\partial y}\right|_n}{1+\delta t \left.\frac{\partial f}{\partial y}\right|_{n+1}}\right| < 1}$$

1^{st} order ODE: Leap-Frog method

- Use central differences approximation for $\frac{d\vec{y}}{dt}$! → $\boxed{\left.\frac{d\vec{y}}{dt}\right|_n \cong \frac{\vec{y}_{n+1}-\vec{y}_{n-1}}{2\delta t}}$

- **Implicit method (2nd order accurate) (aka Stormer-Verlet)**

$$\boxed{\vec{y}_{n+1} = \vec{y}_{n-1} + 2\delta t\vec{f}(t_n,\vec{y}_n) + \mathrm{O}((\delta t)^3)}$$

Stability ? (simpler 1D case) → $\boxed{\delta y_{n+1} = \delta y_{n-1} + 2\delta t \left.\frac{\partial f}{\partial y}\right|_n \delta y_n}$

Algorithm

→ Choose how many time points you want: $\delta t=(t_{end}-t_{begin})/N$

→ Step using Euler or Heun to get y_1 : $\boxed{y_1 = y_0 + \delta t f(t_{begin}, y_0)}$

→ Next iterations performed with: $\boxed{y_{n+1} = y_{n-1} + 2\delta t f(t_n, y_n)}$

1^{st} order ODE: moving up…

- Boosting Euler meant using ***higher accuracy differences*** approximation or using ***implicit schemes***…but there are other options….***Runge-Kutta*** !!!

- But first...more naive approach...expand y(t) in Taylor series

$$\boxed{y_{n+1} \cong y_n + \delta t \left.\frac{dy}{dt}\right|_n + \frac{(\delta t)^2}{2} \left.\frac{d^2 y}{dt^2}\right|_n + \ldots = y_n + \delta t f(t_n, y_n) + \frac{(\delta t)^2}{2} \left(\frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} \boxed{\frac{\partial y}{\partial t}} \right)_n} \Rightarrow f(t_n,y_n)$$

→ Only useful if we can analytically manage it (see *partial derivatives*...)

- ...digging deeper....

$$y_{n+1} \cong y_n + \frac{\delta t}{2} f(t_n, y_n) + \frac{\delta t}{2} \left(f + \delta t \frac{\partial f}{\partial t} + \delta t \frac{\partial f}{\partial y} f(t,y) \right)_n + O(\delta t^3)$$

But also from Taylor series…

$$\boxed{f(t + \delta t, y + \delta t f(t,y)) \cong f(t,y) + \delta t \frac{\partial f}{\partial t} + \delta t \frac{\partial f}{\partial y} f(t,y) + O(\delta t^2)}$$

1^{st} order ODE: Runge-Kutta (2nd order)

- Wrapping up....

$$y_{n+1} \cong y_n + \frac{\delta t}{2} f(t_n, y_n) + \frac{\delta t}{2} \left(f + \delta t \frac{\partial f}{\partial t} + \delta t \frac{\partial f}{\partial y} f(t,y) \right)_n + O(\delta t^3)$$

$$f(t+\delta t, y+\delta t f(t,y)) \cong f(t,y) + \delta t \frac{\partial f}{\partial t} + \delta t \frac{\partial f}{\partial y} f(t,y) + O(\delta t^2)$$

one obtains

$$\boxed{y_{n+1} \cong y_n + \delta t \left[\frac{1}{2} \underbrace{f(t_n, y_n)}_{k_1} + \frac{1}{2} \underbrace{f(t_{n+1}, y_n + \delta t f(t_n, y_n))}_{k_2} \right] + O(\delta t^3)}$$

Heun method again !

➢ It is however not unique ! Expanding in Taylor the expression

$$\boxed{y_{n+1} \cong y_n + \delta t \left[b_1 f(t_n, y_n) + b_2 f(t_n + c_1 \delta t, y_n + \delta t c_2 f(t_n, y_n)) \right]}$$

and comparing term by term with y_{n+1} series →

$$\begin{aligned} & b_1 + b_2 = 1 \\ & c_1 b_2 = 1/2 \ ; \ c_2 b_2 = 1/2 \end{aligned}$$

1st order ODE: Runge-Kutta (2nd order)

- Thus $$\boxed{y_{n+1} \cong y_n + \delta t\left[b_1 f(t_n, y_n) + b_2 f(t_n + c_1\delta t, y_n + \delta t c_2 f(t_n, y_n))\right]}$$

$$b_1 + b_2 = 1$$
$$c_1 b_2 = 1/2 \;\; ; \;\; c_2 b_2 = 1/2$$

Heun method again !

$$\boxed{y_{n+1} \cong y_n + \delta t\left[\frac{1}{2}\underbrace{f(t_n, y_n)}_{k_1} + \frac{1}{2}\underbrace{f(t_{n+1}, y_n + \delta t f(t_n, y_n))}_{k_2}\right] + O(\delta t^3)}$$

Butcher Table

---- y ----

$$t \quad \begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & 1/2 & 1/2 \end{array}$$

$1/2k_1+1/2k_2$

Midpoint method

$$\boxed{y_{n+1} \cong y_n + \delta t \underbrace{f\left(t_{n+1/2}, y_n + \frac{\delta t}{2} f(t_n, y_n)\right)}_{k_2}}$$

Butcher Table

---- y ----

$$t \quad \begin{array}{c|cc} 0 & 0 & 0 \\ 1/2 & 1/2 & 0 \\ \hline & 0 & 1 \end{array}$$

$0k_1+1k_2$

1^{st} order ODE: Runge-Kutta (2nd order)

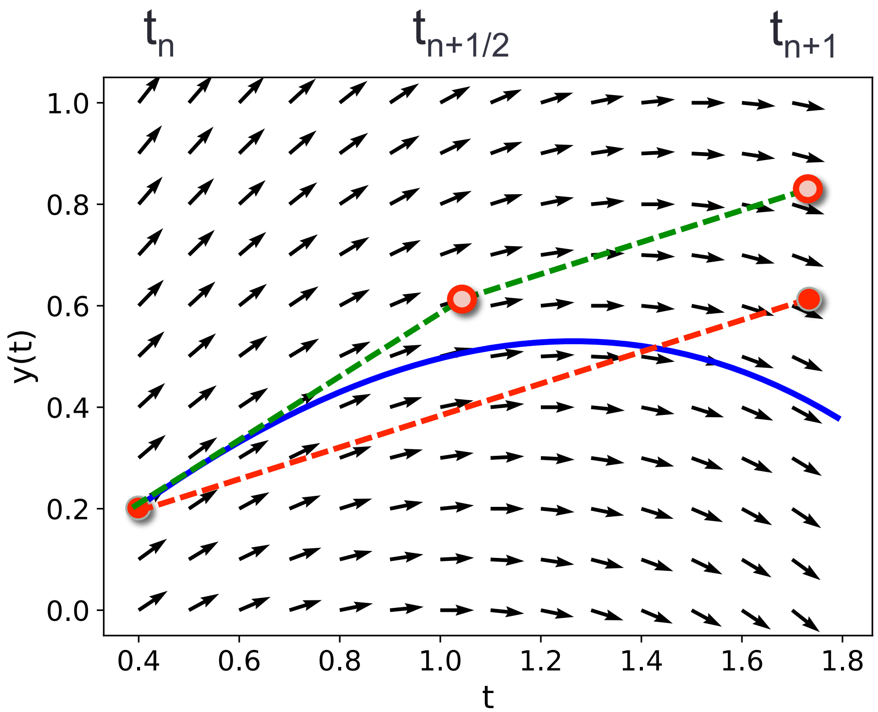
- Graphically…..

$$y_{n+1} \cong y_n + \delta t\left[\underbrace{\frac{1}{2}f(t_n,y_n)}_{k_1} + \underbrace{\frac{1}{2}f(t_{n+1},y_n+\delta t f(t_n,y_n))}_{k_2}\right] + O(\delta t^3)$$

Heun method again !

Midpoint method

$$y_{n+1} \cong y_n + \delta t \underbrace{f\left(t_{n+1/2}, y_n + \frac{\delta t}{2} f(t_n,y_n)\right)}_{k_2}$$



ODE: Y'=0.5Y-T+1

Runge-Kutta algorithm

$$\boxed{\frac{d\vec{y}}{dt} = \vec{f}(t,\vec{y})}$$

→ Multiple time step iterations i.e. RK2 *"iterator"* sequence

```
ODEpoint ODEsolver::RK2_step(const ODEpoint& a, double step){
  int ydim = ...get number of equations dy/dt...
  double * tandY = ...get the set (t,y1,y2,....) from ODEpoint...
  double * Y = ...get the set (y1,y2,....) from ODEpoint...
  double t = ...get the t-value from ODEpoint

  double * K1 = new double [ydim];
  double * K2 = new double [ydim];
  //Calculate K1
  for (int i=0; i< ydim;++i){
    K1[i] = ...step*F[i](tandY)   //symbolic syntax...not the real deal...
  }
  //Calculate K2
  for (int i=0; i< ydim;++i){
    K2[i] = step*F[i](t+step/2,Y[i]+K1[i]/2); // symbolic syntax...
  }
//Now let's advance the y-array...
  tmp = new double [dim]; //the output y-array...
  for (int i=0; i< ydim;++i){
    tmp[i] = Y0[i]+K2[i];
  }
  t += step;
return ODEpoint out(t,tmp,dim);;
}
```

→ ODEpoint is basically an object that stores the (t , $\vec{y}$) numerical solution at a given instant of the ODE.

→ *...later for the possible class declaration of ODEpoint...*

1^{st} order ODE: Runge-Kutta (4th order)

- If one introduces *further intermediate steps*, it becomes similar to a higher order accurate integration e.g. trapezoidal → Simpson !

$$\vec{y}_{n+1} = \vec{y}_n + \int_{t_n}^{t_{n+1}} \vec{f}(t,\vec{y})\,dt \cong \vec{y}_n + \frac{\delta t}{6}\left(\vec{f}(t_n,\vec{y}_n) + 4\vec{f}(t_{n+1/2},\vec{y}_{n+\dots}) + \vec{f}(t_{n+1},\vec{y}_{n+\dots})\right)$$

- In terms of Butcher tables, one increases from 2x2 (e.g. ***midpoint***) to 3x3 (e.g. ***RK3***) or 4x4 (e.g. ***RK4***). As before, several implementations may exist i.e. set of coefficients in expansion $y_{n+1}=y_n+\dots$ not unique !

$$K_1 = \delta t f(t_n, y_n)$$

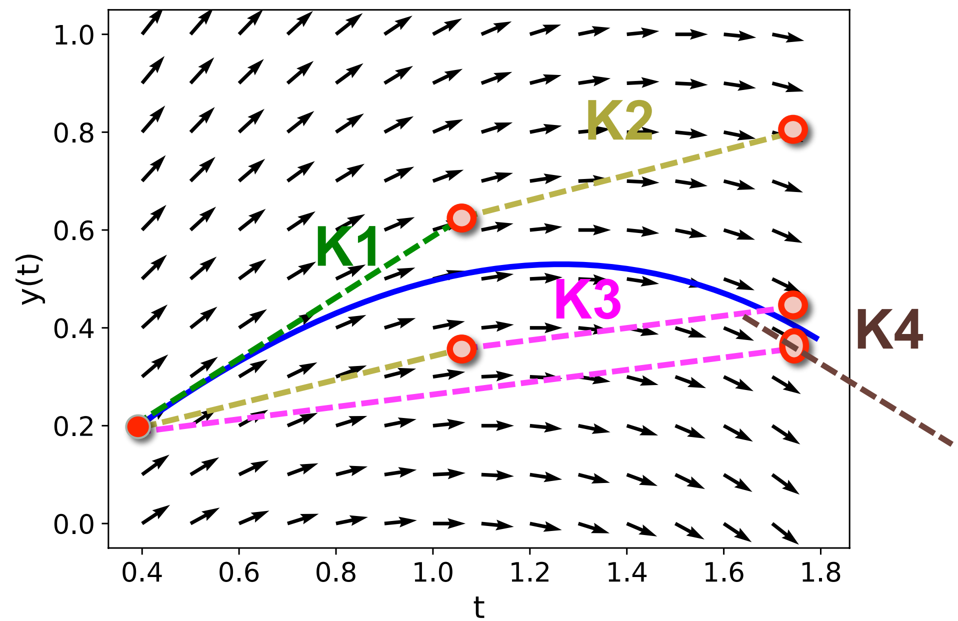
$$K_2 = \delta t f(t_n + \frac{\delta t}{2}, y_n + \frac{K_1}{2})$$

$$K_3 = \delta t f(t_n + \frac{\delta t}{2}, y_n + \frac{K_2}{2})$$

$$K_4 = \delta t f(t_n + \delta t, y_n + K_3)$$

$$y_{n+1} = y_n + \frac{1}{6}\left(K_1 + 2K_2 + 2K_3 + K_4\right) + O((\delta t)^5)$$

Average of 4 different slopes !

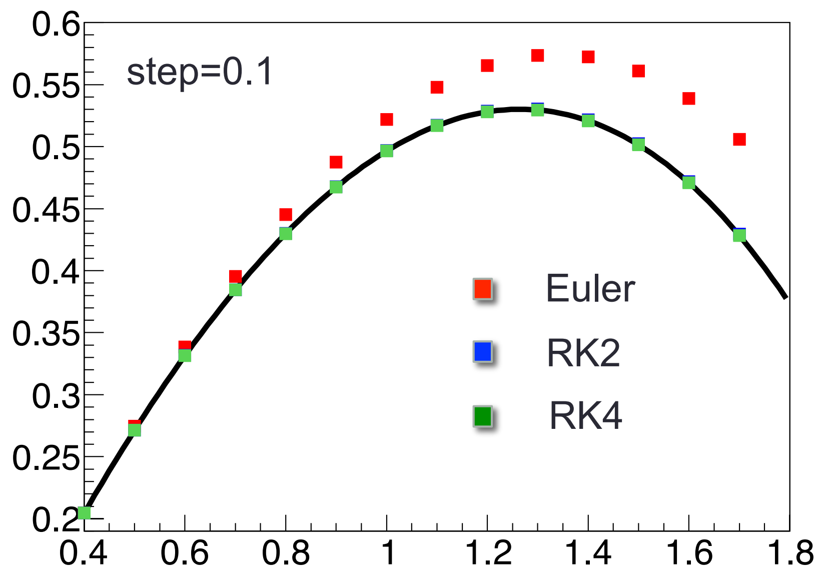


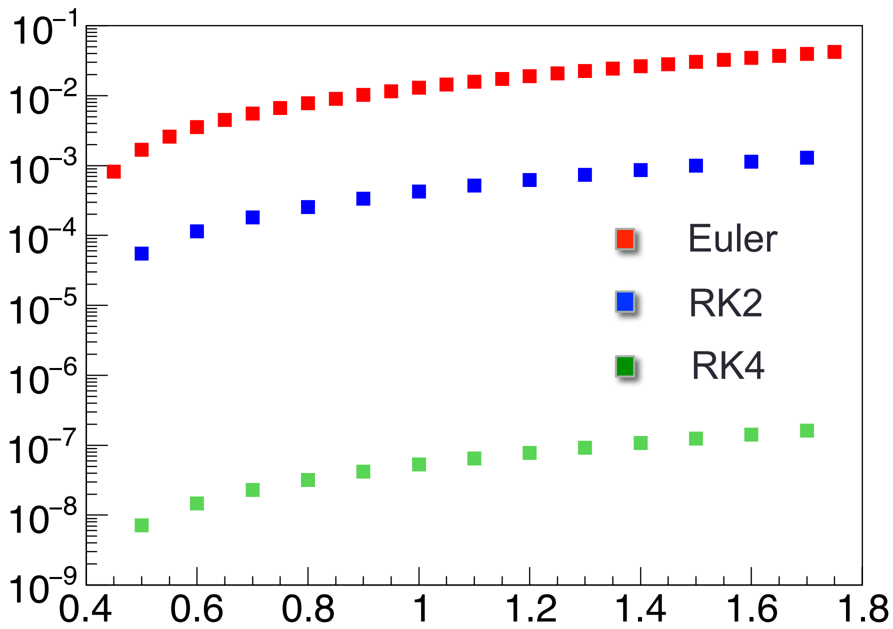
Example: dy/dt = 0.5 * y – t +1

- This ODE has a generic solution

$$y(t) = Ce^{0.5t} + 2t + 2$$

- Use initial value y(t=0.4)=0.2





✓ With ***Euler*** at *step=0.05* still has **2 orders larger** accumulated error than ***RK2*** with step=0.1!

✓ With ***RK4*** one gets accumulated ~**4 orders smaller** error than ***RK2*** for the same step (step=0.1).

Systems of 1st order equations

$$\boxed{\frac{d\vec{y}}{dt} = \vec{f}(t,\vec{y})}$$

→ The same algorithms apply to systems of coupled equations

Lorentz attractor

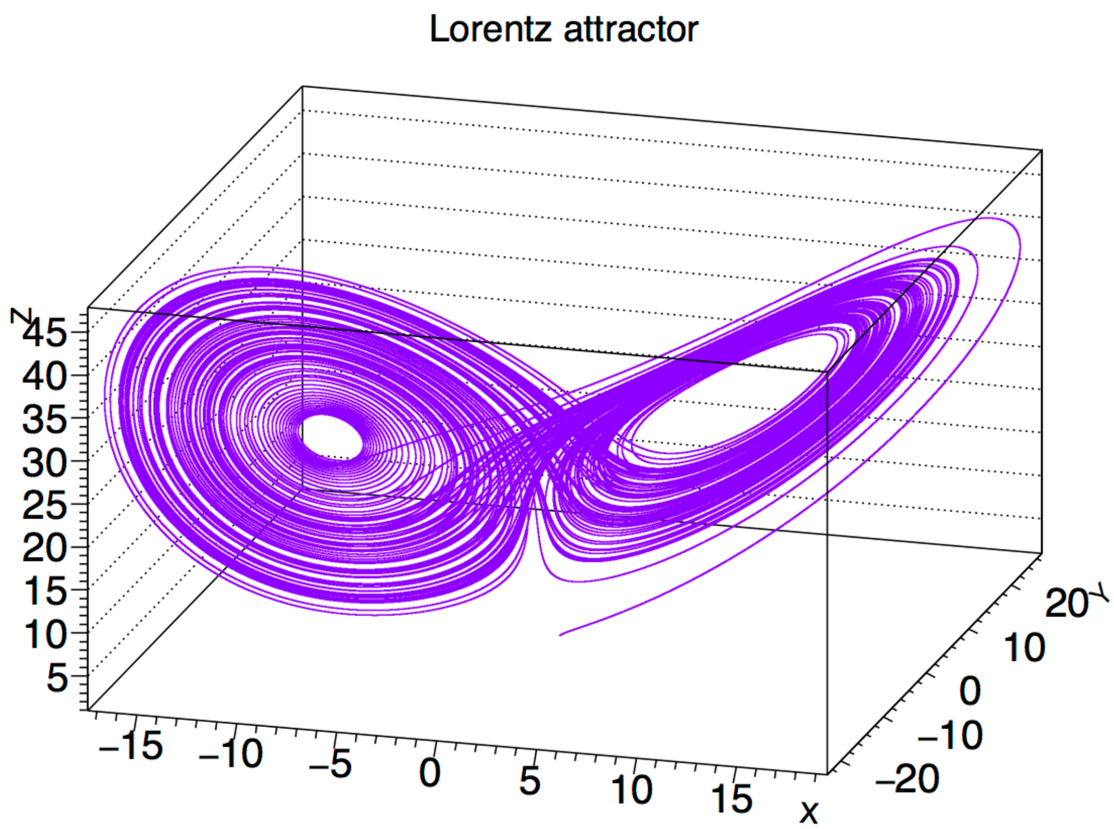
$$\boxed{\begin{aligned} \frac{dx}{dt} &= \sigma(y-x) \\ \frac{dy}{dt} &= x(\rho-z)-y \\ \frac{dz}{dt} &= xy-\beta z \end{aligned}}$$

Parameters

$$\sigma = 10$$

$$\rho = 28$$

$$\beta = 8/3$$



Systems of 1st order equations

Lorentz attractor

$$\frac{dx}{dt} = \sigma(y-x) \qquad \frac{dy}{dt} = x(\rho - z) - y \qquad \frac{dz}{dt} = xy - \beta z$$

$$\downarrow \qquad \downarrow \qquad \downarrow$$

$$\boxed{f^{(0)}(t, y^{(0)}, y^{(1)}, y^{(2)})} \qquad \boxed{f^{(1)}(.......)} \qquad \boxed{f^{(2)}(t, y^{(0)}, y^{(1)}, y^{(2)})}$$

$$\downarrow \qquad \downarrow \qquad \downarrow$$

$$K_1^{(0)} = \delta t f^{(0)}(t_n, \vec{y}_n) \quad K_1^{(1)} = \delta t f^{(1)}(t_n, \vec{y}_n) \quad K_1^{(2)} = \delta t f^{(2)}(t_n, \vec{y}_n)$$

$$K_2^{(0)} = \delta t f^{(0)}(t_n + \frac{\delta t}{2}, \vec{y}_n + \frac{\vec{K}_1}{2}) \quad \ldots \quad K_2^{(2)} = \delta t f^{(2)}(t_n + \frac{\delta t}{2}, \vec{y}_n + \frac{\vec{K}_1}{2})$$

$$K_3^{(0)} = \delta t f^{(0)}(t_n + \frac{\delta t}{2}, \vec{y}_n + \frac{\vec{K}_2}{2}) \quad \ldots \quad K_3^{(2)} = \delta t f^{(2)}(t_n + \frac{\delta t}{2}, \vec{y}_n + \frac{\vec{K}_2}{2})$$

$$K_4^{(0)} = \delta t f^{(0)}(t_n + \delta t, \vec{y}_n + \vec{K}_3) \quad \ldots \quad K_4^{(2)} = \delta t f^{(2)}(t_n + \delta t, \vec{y}_n + \vec{K}_3)$$

$$\vec{y}_{n+1} = \vec{y}_n + \frac{1}{6}\left(\vec{K}_1 + 2\vec{K}_2 + 2\vec{K}_3 + \vec{K}_4\right)$$

Single equation...

$$K_1 = \delta t f(t_n, y_n)$$

$$K_2 = \delta t f(t_n + \frac{\delta t}{2}, y_n + \frac{K_1}{2})$$

$$K_3 = \delta t f(t_n + \frac{\delta t}{2}, y_n + \frac{K_2}{2})$$

$$K_4 = \delta t f(t_n + \delta t, y_n + K_3)$$

$$y_{n+1} = y_n + \frac{1}{6}\left(K_1 + 2K_2 + 2K_3 + K_4\right) + O((\delta t)^5)$$

ODEpoint Class

```
class ODEpoint {
public:
  ODEpoint();//default constructor
  ~ODEpoint();//destructor
  ODEpoint(double tval, double* funct, int Ndimf); //using double *
  ODEpoint(double tval, vector<double> funct); //using vector<double>
  ODEpoint(const ODEpoint&); //copy constructor

  //member access functions
  vector<double> Get_Var_vec() const; //return the y1,…,yNdim dependent variables
  double * Get_Var_ptr() const; //same but as double *
  double * Get_VarTime() const; //first the y1,…,yNdim then t
  int GetNdim() const; //return the number of dependent variables
  double Get_Time() const; //return only the independent variable (t)

  void Set_Time(double tval) {t = tval;} //Set dependent variable
  void Set_Var(vector<double> funct) {var = funct; Ndim = var.size();}

  //operators
   ODEpoint & operator=(const ODEpoint& P);
  const double& operator[](int i) const {return var[i];}
  double& operator[] (int i) {return var[i];}

  void Print() const;

private:
  double t;
  vector<double> var;
  int Ndim;
};
```

$$\boxed{\frac{d\vec{y}}{dt} = \vec{f}(t,\vec{y})}$$

- → Each ODEpoint represents the argument of the function on rhs
- → Since ODEpoint is valid for systems of ODE, Ndim is needed to refer to number of ODEs...
- → Ordering (t,y) or (y,t) is of taste/practicality

ODEsolver Class

```
class ODEsolver{
public:
  ODEsolver(vector<TFormula> FF);
  ~ODEsolver();

vector <ODEpoint> Eulersolver(const ODEpoint& P0, double xmin,
double xmax, double h_step);
vector<ODEpoint> RK2solver(const ODEpoint& P0, double xmin, double
xmax, double h_step);
vector <ODEpoint> RK4solver(const ODEpoint& P0, double xmin, double
xmax, double h_step);
  vector <ODEpoint> RK4_AdapStep(const ODEpoint& P0, double xmin,
double xmax, double h_step);
  vector<ODEpoint> Heun(const ODEpoint& P0, double xmin, double
xmax, double h_step);
  ...
  void SetODEfunc(vector<TFormula> FF);
private:
  ODEpoint Heun_iterator (const ODEpoint&, double step);
  ODEpoint EULER_iterator (const ODEpoint&, double step);
  ODEpoint RK2_iterator (const ODEpoint&, double step);
  ODEpoint RK4_AS_iterator(const ODEpoint&, double step, vector <
vector <double> >& K);
  ODEpoint RK4_iterator(const ODEpoint&, double step);
  vector<TFormula> F;
};
```

$$\boxed{\frac{d\vec{y}}{dt} = \vec{f}(t,\vec{y})}$$

→ Solver defines whole time range and calls a single step iterator…

→ Each iterator advances only one time step.

→ Multiple solvers available and easily extended to additional ones…

High-order equations

$$\frac{d^2x}{dt^2} = g(t,x,\frac{dx}{dt}) \longrightarrow \frac{d^2x}{dt^2} + \frac{\lambda}{m}\frac{dx}{dt} + \frac{k}{m}x = 0$$

Damped harmonic oscillator

…is the same as

$$\begin{cases} \dfrac{dx}{dt} = y \\ \dfrac{dy}{dt} = g(t,x,y) \end{cases} \Leftrightarrow \begin{cases} \dfrac{dy^{(0)}}{dt} = f^{(1)}(t,y^{(0)},y^{(1)}) \\ \dfrac{dy^{(1)}}{dt} = f^{(2)}(t,y^{(0)},y^{(1)}) \end{cases} \Leftrightarrow \begin{bmatrix} \dfrac{dy^{(0)}}{dt} \\ \dfrac{dy^{(1)}}{dt} \end{bmatrix} = \begin{bmatrix} f^{(1)}(t,y^{(0)},y^{(1)}) \\ f^{(2)}(t,y^{(0)},y^{(1)}) \end{bmatrix}$$

Newton's 2nd…

$$(x,y,z,v_x,v_y,v_z) \to (y^{(0)},y^{(1)},y^{(2)},y^{(3)},y^{(4)},y^{(5)})$$

$$m\frac{d^2\vec{x}}{dt^2} = \vec{F}(t,\vec{x},\dot{\vec{x}})$$

$$\begin{cases} \dfrac{d\vec{x}}{dt} = \vec{v} \\ \dfrac{d\vec{v}}{dt} = \dfrac{\vec{F}(t,\vec{x},\vec{v})}{m} \end{cases}$$

$$\boxed{\begin{array}{ll} f^{(0)}(t_n,\vec{y}_n) = y^{(3)} & f^{(3)}(t_n,\vec{y}_n) = \dfrac{F_x(t_n,\vec{y}_n)}{m} \\ f^{(1)}(t_n,\vec{y}_n) = y^{(4)} & f^{(4)}(t_n,\vec{y}_n) = \dfrac{F_y(t_n,\vec{y}_n)}{m} \\ f^{(2)}(t_n,\vec{y}_n) = y^{(5)} & f^{(5)}(t_n,\vec{y}_n) = \dfrac{F_z(t_n,\vec{y}_n)}{m} \end{array}}$$

$$\boxed{\vec{y}_{n+1} = \vec{y}_n + \frac{1}{6}\left(\vec{K}_1 + 2\vec{K}_2 + 2\vec{K}_3 + \vec{K}_4\right)}$$

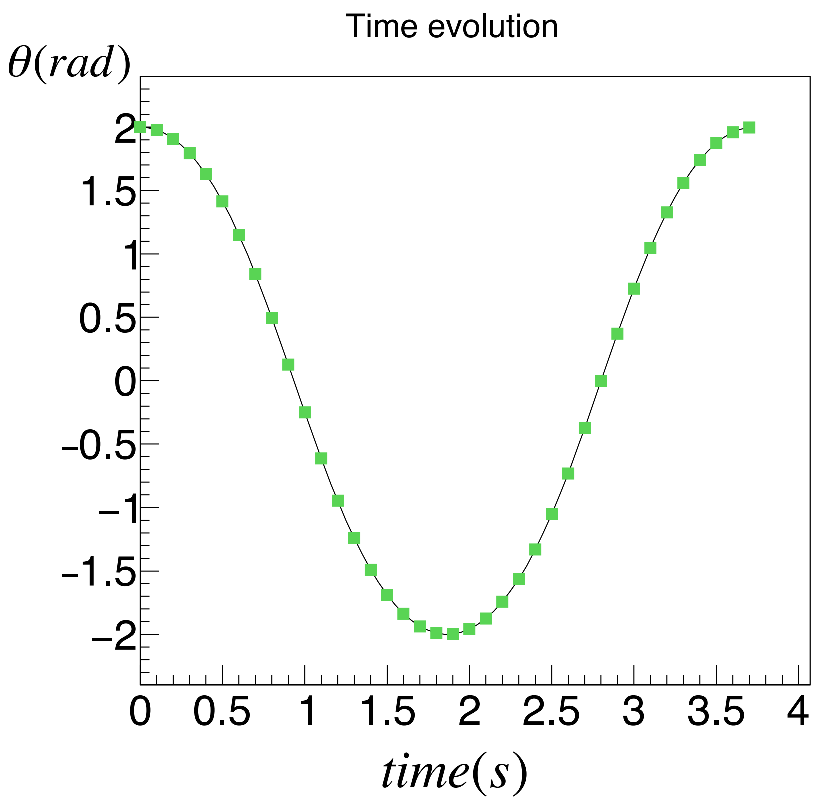
RK4 rulez…

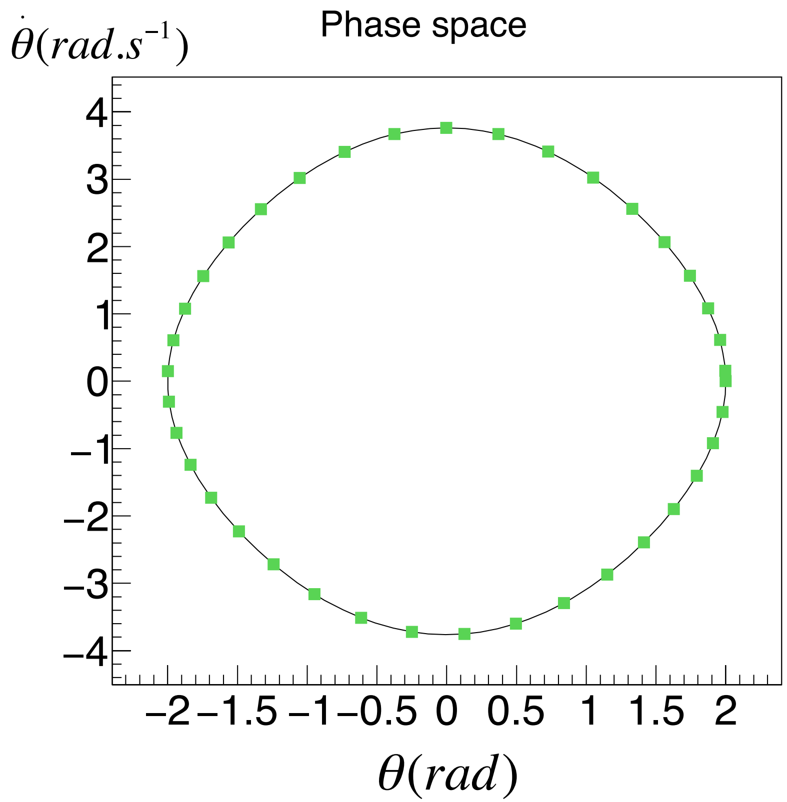
Example: pendulum…

$$\frac{d^2\theta}{dt^2} = -k\sin(\theta) \quad , k = g/L$$

$$\theta(0) = \theta_0 \quad , \quad \omega(0) = \omega_0$$

$$\begin{cases} \dfrac{d\theta}{dt} = \omega \\ \dfrac{d\omega}{dt} = -k\sin(\theta) \end{cases} \Leftrightarrow \begin{cases} \dfrac{dy^{(0)}}{dt} = f^{(1)}(t, y^{(0)}, y^{(1)}) = y^{(1)} \\ \dfrac{dy^{(1)}}{dt} = f^{(2)}(t, y^{(0)}, y^{(1)}) = -k\sin(y^{(0)}) \end{cases}$$





2^{nd} order ODE: 2^{nd} derivative appr.

$$\frac{d^2\theta}{dt^2} = -k\sin(\theta) \quad , k = g/L$$

$$\theta(0) = \theta_0 \quad , \quad \omega(0) = \omega_0$$

$$\boxed{\frac{y_{n+1} - 2y_n + y_{n-1}}{(\delta t)^2} \cong \left.\frac{d^2 y}{dt^2}\right|_n = f(t_n, y_n)}$$

Verlet algorithm

- How to.......
  - Time step: δt
  - Initial condition at $t=t_0$ (2 needed) : $y(t_0) = y_0$, $\left.\frac{dy}{dt}\right|_{t_0} = \left(\frac{dy}{dt}\right)_0$
  - First iteration: $y_1 = y_0 + \delta t \left(\frac{dy}{dt}\right)_0 + \frac{(\delta t)^2}{2} f(t_0, y_0)$
  - Following iterations: $$\boxed{\begin{aligned} y_{n+1} &= 2y_n - y_{n-1} + (\delta t)^2 f(t_n, y_n) \\ t_{n+1} &= t_n + \delta t \end{aligned}}$$

2^{nd} order ODE: revisiting the pendulum...

$$\begin{cases} \dfrac{dy^{(0)}}{dt} = f^{(1)}(t, y^{(0)}, y^{(1)}) = y^{(1)} \\ \dfrac{dy^{(1)}}{dt} = f^{(2)}(t, y^{(0)}, y^{(1)}) = -k\sin(y^{(0)}) \end{cases}$$

$$y^{(0)}(0) = \theta_0 \quad , \quad y^{(1)}(0) = \omega_0$$

- Euler method

$$\begin{cases} y_{n+1}^{(0)} = y_n^{(0)} + \delta t y_n^{(1)} \\ y_{n+1}^{(1)} = y_n^{(1)} + \delta t\left(-k\sin(y_n^{(0)})\right) \end{cases}$$

- Euler-Cromer method

$$\begin{cases} y_{n+1}^{(0)} = y_n^{(0)} + \delta t \boxed{y_{n+1}^{(1)}} \\ y_{n+1}^{(1)} = y_n^{(1)} + \delta t\left(-k\sin(y_n^{(0)})\right) \end{cases}$$

- Euler-Verlet method

$$\begin{cases} y_{n+1}^{(0)} = 2y_n^{(0)} - y_{n-1}^{(0)} + (\delta t)^2\left(-k\sin(y_n^{(0)})\right) \\ y_n^{(1)} = \dfrac{y_{n+1}^{(0)} - y_{n-1}^{(0)}}{2\delta t} \end{cases}$$

(bonus !)

Initial condition at $t=t_0$

$$y_1^{(0)} = y^{(0)}(t_0) + \delta t y^{(1)}(t_0) + \frac{(\delta t)^2}{2}\left(-k\sin(y^{(0)}(t_0))\right)$$

$$y^{(1)}(t_0) \quad - \quad given$$

COMPUTATIONAL PHYSICS

Numerical methods

Ordinary Differential Equations

- ✓ *Adaptive methods*

Adaptive stepping in RK methods

- When using RK methods of order-**p** (p terms in K-sum), it is known that the error per time step scales with $(\delta t)^{p+1}$. Time stepping N steps means also $N=(t_f-t_i)/\delta t$ -----> **Total error ~ $(\delta t)^{p+1}$**.
- What the the *optimal time step* to keep the error below given tolerance ?
- How can we *adapt* this time step at every iteration ?

- ***Two common ways***
  - Step doubling
  - RK45 (Runge-Kutta-Fehlberg)

Step doubling in RK methods

Step doubling

- Basic idea: Compare single time step (**h**) and twice half step (**h/2**)

$$\tilde{y}_n = y_n^{(\text{single})} + ch^{p+1} + O(h^{p+2})$$

$$\tilde{y}_n = y_n^{(\text{double})} + \boxed{2c\left(\frac{h}{2}\right)^{p+1}} + O(h^{p+2})$$

$$\Longrightarrow \quad \tilde{y}_n = y_n^{(\text{double})} + \underbrace{\boxed{\frac{\left|y_n^{(\text{single})} - y_n^{(double)}\right|}{2^p - 1}}}_{\varepsilon} + O(h^{p+2})$$

Subtract…

$$\boxed{c = \frac{\left|y_n^{(\text{single})} - y_n^{(\text{double})}\right|}{\left(1-2^{-p}\right)h^{p+1}}}$$

$$\boxed{h_{new} = \beta h\left(\frac{\varepsilon_0}{\varepsilon}\right)^{1/p+1}}$$

ε_0 – desired tolerance
β – safety margin (~0.9)

- *If $\varepsilon > \varepsilon_0$ repeat step analysis (t_n not stepped) with reduced $h = h_{new}$*
- *If $\varepsilon < \varepsilon_0$ advance $t_n + h$ and set increased $h = h_{new}$*

RK-Fehlberg

- ***Runge-Kutta-Fehlberg** (with Cash-Karp coefficients)*
  - Basic idea: Compare two different approximations (**4th** order and 5th order).

$$\varepsilon = \left| y_n^{(fourth)} - y_n^{(\mathrm{fifth})} \right| = \left| \sum_{i=1}^{6} \left(c_i^{(fourth)} - \mathrm{c}_i^{(\mathrm{fifth})} \right) K_i \right| \propto h^5$$

c – RK expansion constants

and

$$\boxed{K_i = hf\left(t + A_i, y_n + \sum_{l=1}^{6} B_i^{(l)} K_l \right)}$$

A_i	$B_i^{(l)}$					
0						
1/5	1/5					
3/10	3/40	9/40				
3/5	3/10	–9/10	6/5			
1	–11/54	5/2	–70/27	35/27		
7/8	1631/55296	175/512	575/13824	44275/110592	253/4096	
	37/378	0	250/621	125/594	0	512/1771
	2825/27648	0	18575/48384	13525/55296	277/14336	1/4

Cash and Karp coefficients

$c_i^{(fifth)}$

$c_i^{(fourth)}$

Time stepping in adaptive schemes

❑ *Strategy and implementation*

- Consider a system of ***m*** 1st order ODEs.
- Calculate the error estimate e.g. from the the 4th and 5th order estimates

$$\varepsilon^{(j)}(h) = \sum_{i=1}^{6}\left(c_i^{(fourth)} - \mathrm{c}_i^{(\mathrm{fifth})}\right)K_i^{(j)}(h) \qquad j=1..m$$

- Global *vs* *worst-offender*
  - The m-equations may yield solutions of different magnitude ! Relative *vs* absolute
  - If worst-offender rules then the rescaling of step-h obeys needs of stringiest Eq.

Global

$$h_{new} = \beta h\left(\frac{\varepsilon_0}{\varepsilon}\right)^{1/5} \qquad \varepsilon = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(\varepsilon^{(j)}(h)\right)^2}$$

→ Depending on $\varepsilon>\varepsilon_0$ or $\varepsilon<\varepsilon_0$ proceed as step doubling

Worst-offender

$$h_{new} = \beta h\left(\frac{\varepsilon_0}{\varepsilon}\right)^{1/5} \qquad \varepsilon = Max\left(\left|\frac{\varepsilon^{(j)}(h)}{\mathrm{Ref}}\right|\right)$$

Ref = $y_n^{(j)}$ or Ref$^{(j)}$

COMPUTATIONAL PHYSICS

Numerical methods

Ordinary Differential Equations

✓ *Boundary value problems*

Boundary value problems (*BVP*)

- ***What and Why***

- ODE methods such as RK are “initial value based” methods.
- Physics on the other hand calls sometimes for boundary value problems !

e.g.

- Heat conducting rod in steady state from two reservoirs at different temperatures $T(x = 0) = T_0$ and $T(x = L) = T_L$

$$\boxed{0 = \frac{k}{C_p \rho} \frac{\partial^2 T}{\partial x^2}}$$

C_p – specific heat capacity [J/(Kg.K)]
k – Thermal conductivity
ρ – density of the rod’s material

- Projectile launched vertically with unknown velocity from y=0 and reaching a maximum height y(t=10s)=50m.

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

Y(0)=0
Y(5)=40

Y’(0)=???

Boundary value problems

- ***…How…?***

- Assume one has a ***initial value problem*** (IVP) using a guess initial value e.g. initial velocity, and apply as usual an integrator like RK4.
- Depending on the value reach at end of run, *iteratively change the initial value guess* until the target boundary value is reached….*simple* ?

$$\frac{d^2y}{dt^2} = f(t,y,y') \qquad Y(a)=\alpha \quad Y(b)=\beta$$

$$\frac{d^2y}{dt^2} = f(t,y,y') \qquad Y(a)=\alpha \quad Y'(a)=u$$

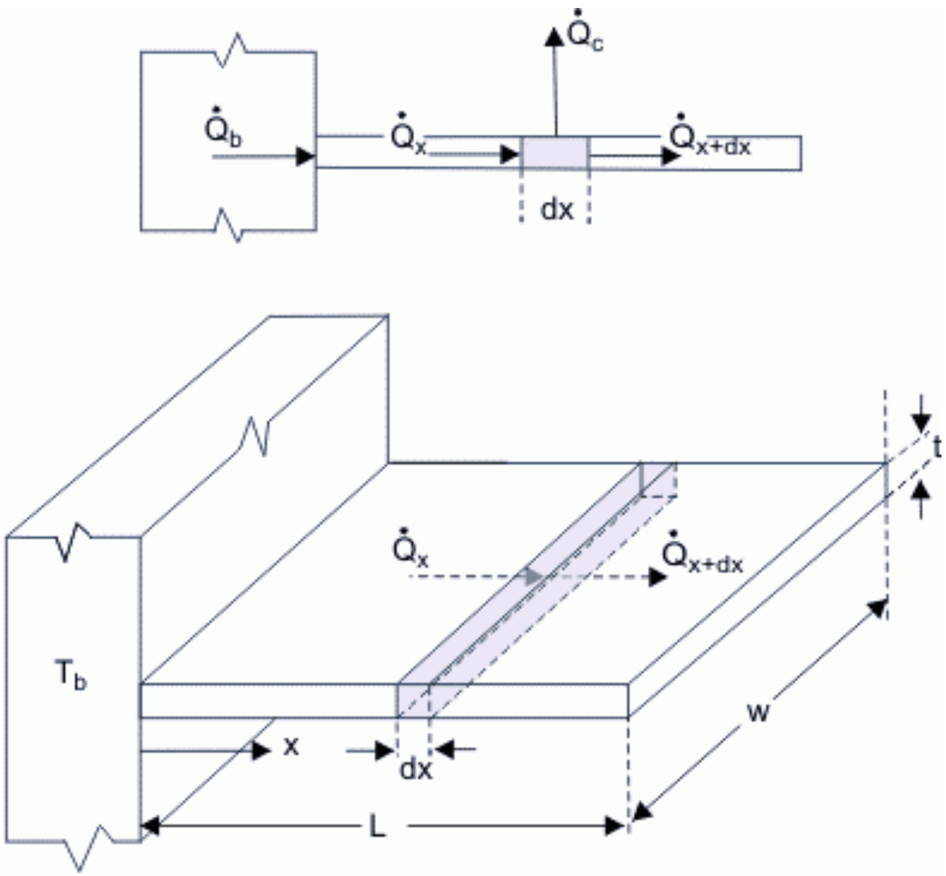
Initial value guess

- Depending on u, one gets ***Y(b)=g(u)***, with ***g*** unknown.
- Define $F(u)=g(u)-\beta$ and search for $F(u)=0$! How ?
- Choose two value u_0 and u_1 such that $F(u_0)F(u_1)<0$.
- Now use root find methods e.g. secant method

$$u_{i+1} = u_i - F(u_i)\frac{u_i - u_{i-1}}{F(u_i) - F(u_{i-1})}$$

Boundary value problems - example

❑ *Heat dissipation from plain fin*



http://www.thermopedia.com/content/671/

- Heat flux density: $q = -k\dfrac{\partial T}{\partial x}$ k-thermal conductivity
- Heat power balance $\dot{Q}_x = \dot{Q}_{x+dx} + \dot{Q}_{\text{sink}}$ with

  $\dot{Q}_x = -kA\dfrac{\partial T}{\partial x}$ A – cross-section area
- On the sink $\dot{Q}_{\text{sink}} = hPdx(T - T_\infty)$

  P=2(t+w) ; Pdx – elementary area to sink

  h – heat-transfer coefficient
- Taylor series $\dot{Q}_{x+dx} = \dot{Q}_x + \dfrac{d\dot{Q}_x}{dx}dx$
- Finally $$\frac{d\dot{Q}_x}{dx}dx + hPdx(T - T_\infty) = 0$$

  or $$\boxed{\frac{d^2T}{dx^2} - \frac{hP}{kA}(T - T_\infty) = 0}$$

Heat dissipation – shooting method

- Assumption: vanishing T_∞ and some scaled temperature

$$\boxed{\frac{d^2y}{dx^2} = 2y} \qquad \begin{cases} y(x=0)=1 \\ y'(x=1)=0 \end{cases}$$ ***(zero heat flux)***

- Analytical solution:

$$y(x) = c_1 e^{\sqrt{2}x} + c_2 e^{-\sqrt{2}x} \qquad \begin{cases} c_1 = 0.05581 \\ c_2 = 0.94419 \end{cases}$$

- Using shooting method, first get a system of ODEs

$$\begin{cases} \dfrac{dy}{dx} = z \\ \dfrac{dz}{dx} = 2y \end{cases}$$ ***with*** $$\begin{cases} y(x=0)=1 \\ z(x=1)=0 \end{cases}$$

- For ODEs we need initial b.c. so take an educated guess….

$$\begin{cases} y(x=0)=1 \\ z(x=0)=u_0 \end{cases}$$

- Now time advance the solution up to x=1 using e.g. RK4 to obtain $g(u_0)=z^{(0)}(1)$.
- Check signal of $F(u_0)=z(1)-g(u_0)$
- Repeat with another initial b.c.

$$\begin{cases} y(x=0)=1 \\ z(x=0)=u_1 \end{cases}$$

such that, after time stepping to x=1, one gets $F(u_1)=z(1)-g(u_1)$ has opposite sign of $F(u_0)$

- Now proceed with usual root finding methods to search for F(u)~0 !

Shooting method - example

- Assumption: vanishing T_∞ and some scaled temperature

$$\boxed{\frac{d^2y}{dx^2} = 2y} \qquad \begin{cases} y(x=0) = 1 \\ y'(x=1) = 0 \end{cases} \quad \textit{(zero heat flux)}$$

- Initial guesses:

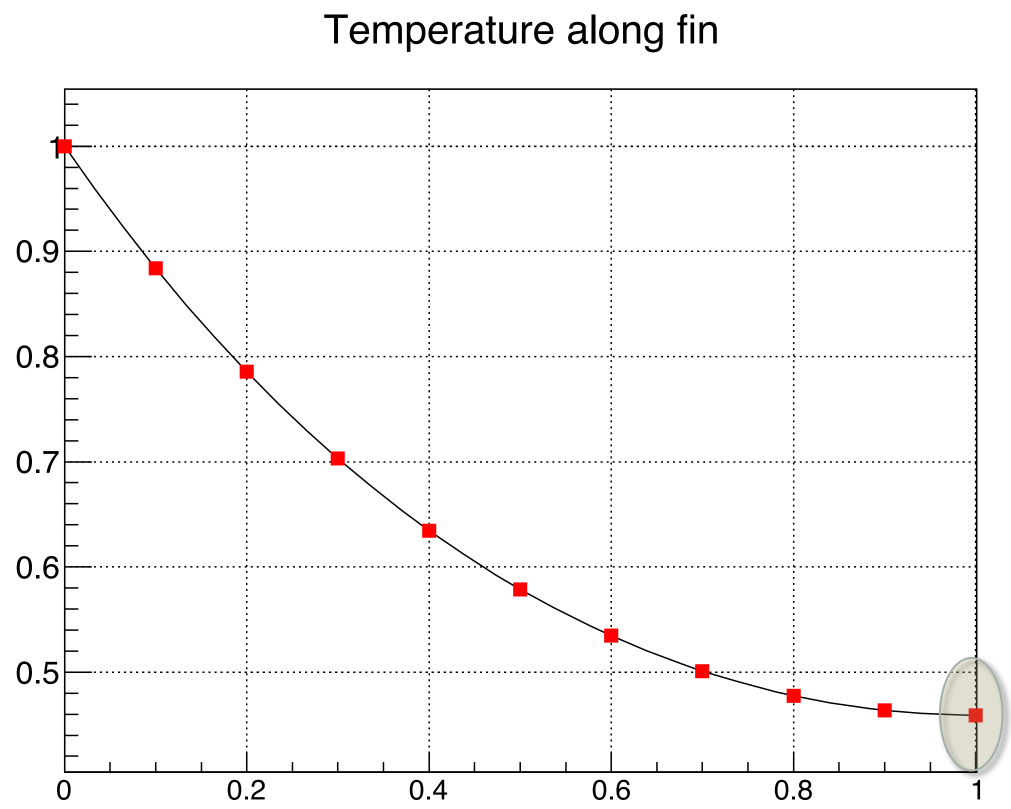
$$z(0) = 0 \quad z(0) = -2$$

- This yields

z(0)=0 → $z^{(guess1)}(1)$ = 2.73658
z(0)=-2 → $z^{(guess1)}(1)$ = -1.61977

- Using a single secant step…

z(0)=-1.25637 → $z^{(guess1)}(1)$ = 2.775e-16

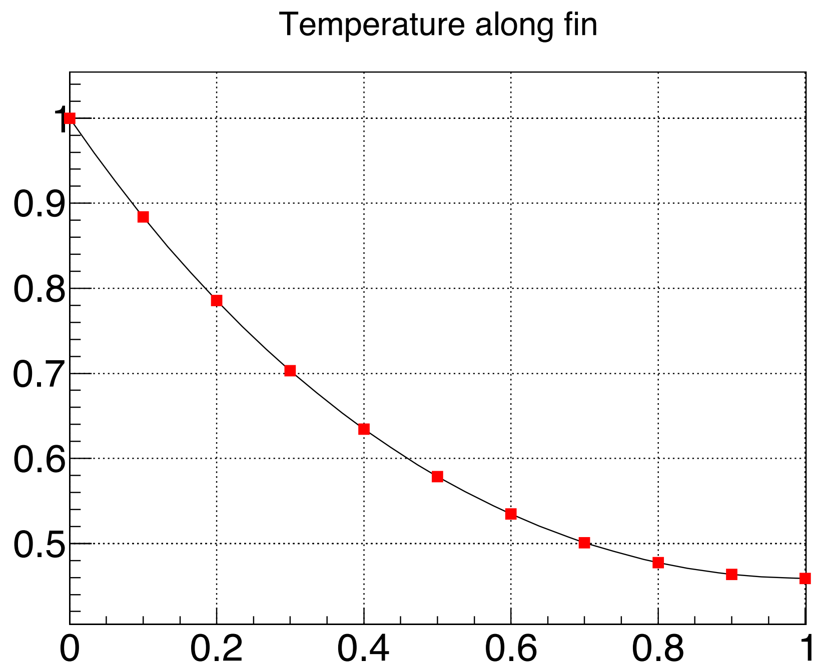


BVP with finite differences…

- ***Basic Idea***: instead of "guessing" how to reach a forward boundary condition, use it while discretizing the solution domain…

- Yes...it implies solving a ***system of linear equations***...

$$\boxed{\frac{d^2y}{dx^2} = 2y} \qquad \begin{cases} y(x=0)=1 \\ y'(x=1)=0 \end{cases}$$



Recall that

$$f'(x_i) = \frac{f(x_i+h) - f(x_i-h)}{2h}$$

$$f''(x_i) = \frac{f(x_i+h) - 2f(x_i) + f(x_i-h)}{h^2}$$

$$\Rightarrow \boxed{\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} = 2y_i}$$

with i=0..N → What about the boundary conditions ?

BVP with finite differences (cont)

- The system of equations is undetermined if not using the bound. cond. !

$$\begin{cases} \dfrac{1}{h^2}y_0 + \left(-\dfrac{2}{h^2} - 2\right)y_1 + \dfrac{1}{h^2}y_2 = 0 \\ \dots \\ \dfrac{1}{h^2}y_{N-2} + \left(-\dfrac{2}{h^2} - 2\right)y_{N-1} + \dfrac{1}{h^2}y_N = 0 \end{cases}$$

- (N-1) equations
- (N+1) variables

2 Extra Eq.

$$\Rightarrow \begin{cases} y(x=0) = 1 \\ y'(x=1) = 0 \end{cases}$$

$$y_0 = 1 \qquad \frac{y_{N+1} - y_{N-1}}{2h} = 0$$

Ghost point !

- In general, one may find

$$\begin{cases} y(x=a) = \alpha \\ y(x=b) = \beta \end{cases} \Rightarrow \begin{cases} y_0 = \alpha \\ y_N = \beta \end{cases} \qquad \begin{cases} y'(x=a) = \alpha \\ y'(x=b) = \beta \end{cases} \Rightarrow \begin{cases} \dfrac{y_1 - y_{-1}}{2h} = \alpha \\ \dfrac{y_{N+1} - y_{N-1}}{2h} = \beta \end{cases}$$

BVP with finite differences (cont)

- The system of equations is now perfectly solvable...

$$\begin{cases} \dfrac{1}{h^2}1+\left(-\dfrac{2}{h^2}-2\right)y_1+\dfrac{1}{h^2}y_2=0 \\ \cdots \\ \dfrac{1}{h^2}y_{N-2}+\left(-\dfrac{2}{h^2}-2\right)y_{N-1}+\dfrac{1}{h^2}y_N=0 \\ \dfrac{1}{h^2}y_{N-1}+\left(-\dfrac{2}{h^2}-2\right)y_N+\dfrac{1}{h^2}y_{N-1}=0 \end{cases}$$

$$\Leftarrow \boxed{y_0=1}$$

$$\Leftarrow \boxed{\dfrac{y_{N+1}-y_{N-1}}{2h}=0}$$

Ghost point
→
New Equation

- Proceed to Tridiagonal solver (*or in general G.Elim.*) to solve the system. The number of points in the interval [0,1] is a user's choice (or *discretization needs*).

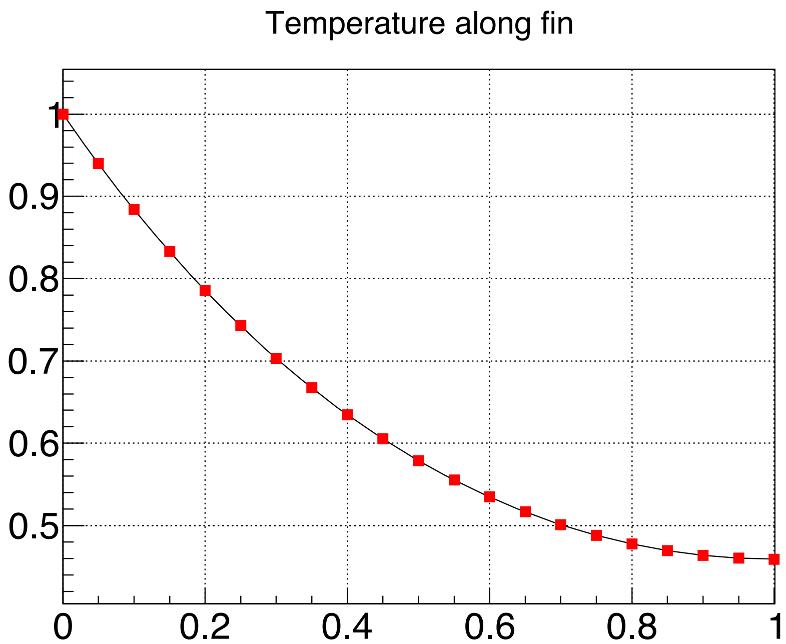
BVP with finite differences (cont)

- Matricial form and solution

$$\left[\begin{array}{cccccc|c} -2(1+h^2) & 1 & 0 & \vdots & 0 & -1 \\ 1 & -2(1+h^2) & 1 & \vdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & 1 & -2(1+h^2) & 1 & 0 \\ 0 & 0 & 0 & 2 & -2(1+h^2) & 0 \end{array}\right]$$

B.C. at x=a

B.C. at x=b



Note:

- The solution is easily "contaminated" by boundary conditions with lower order of accuracy
- Try using forward/backward derivative at x=1….