Modelling in Computational Science, BERN01, 7.5hp

Practical and theoretical knowledge of numerical methods used for solving ODE modells for real Life Science problems.

1st study period, autumn semester

Ordinary Differential Equations – Initial Value Problems

A system of first order Ordinary Differential Equations (ODES)

$$\begin{cases} \dfrac{dy_1}{dt} = f_1(t, y_1, y_2, \dots, y_N) \\ \quad\vdots \\ \dfrac{dy_N}{dt} = f_N(t, y_1, y_2, \dots, y_N) \end{cases}$$

In vector form $\dfrac{dY}{dt} = f\ (t, Y), \qquad Y(t_0) = Y_o$

Ordinary Differential Equations – Initial Value Problems

Looking only at first-order ODEs is not a strong limitation, because any Nth order ODE can be written as a system of first order ODE

$$\frac{d^N y}{dt^N} = f\left(t, y, \frac{dy}{dt}, .., \frac{d^{N-1}y}{dt^{N-1}}\right)$$

We make notation

$$\begin{aligned} y_1 &= y \\ y_2 &= \frac{dy}{dt} \\ &\dots \\ y_N &= \frac{d^{N-1}y}{dt^{N-1}} \end{aligned}$$

thus

$$\begin{cases} \dfrac{dy_1}{dt} = y_2 \\ \dfrac{dy_2}{dt} = y_3 \\ \dots \\ \dfrac{dy_N}{dt} = f(t, y_1, .., y_N) \end{cases}$$

Example

$$\frac{d^3 y}{dt^3} = t^2$$

$$\begin{aligned} y_1 &= y \\ y_2 &= \frac{dy}{dt} \\ y_3 &= \frac{d^2 y}{dt^2} \end{aligned}$$

thus

$$\begin{cases} \dfrac{dy_1}{dt} = y_2 \\ \dfrac{dy_2}{dt} = y_3 \\ \dfrac{dy_3}{dt} = t^2 \end{cases}$$

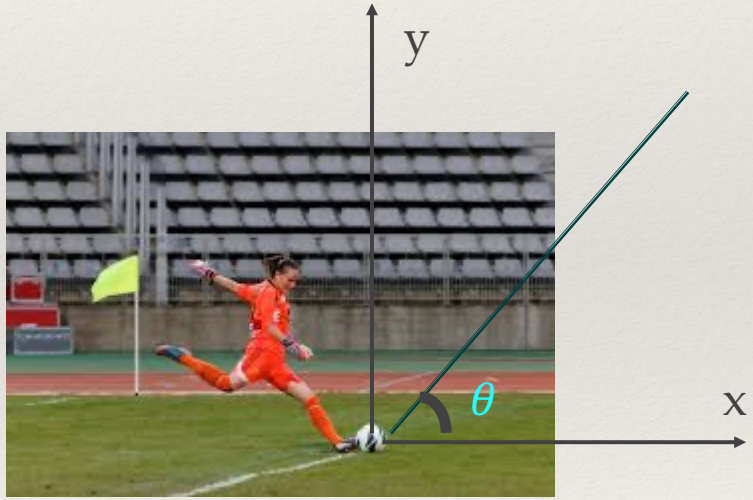
Ordinary Differential Equations - ODE

An example – Kicking a footbal in the air

A ball of mass m is kicked at an angle θ
with an initial velocity $v_0 = (v_{x,0}, v_{y,0})$
in a gravitational field with gravitational constant g

Newton equations of motions

$$m\frac{d^2x}{dt^2} = F_{(drag,x)}$$
$$m\frac{d^2y}{dt^2} = -mg + F_{(drag,y)}$$



F_{drag} is present because of the air

F_{drag} the complicated part of the model

Kicking a footbal in the air

$$F_{drag} = -(B_1 v + B_2 v^2)V$$

where

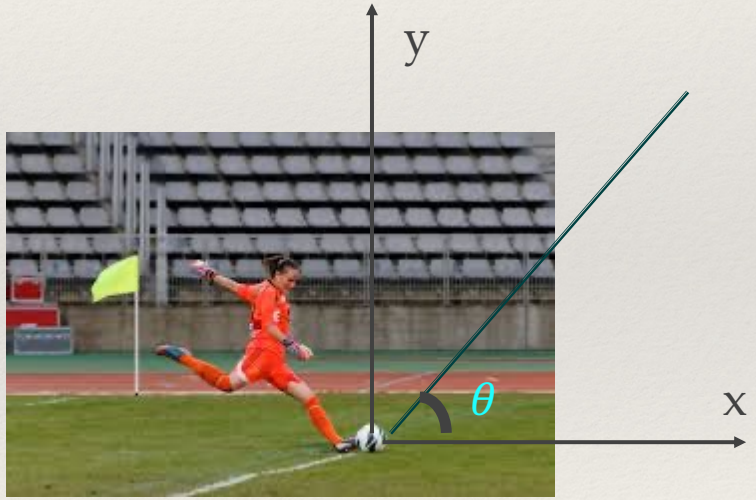
$v = v_x^2 + v_y^2$ with $v_{x,y}$ being the x,y- component of the velocity,
$V = (v_x X + v_y Y)/v$ is a unit vector in the direction of the velocity.

$$B_1 = 6\pi\eta R$$

Stoke's drag - for a sphere
R is ball radius and η the air viscosity

$$B_2 = \frac{1}{2} C\rho A$$

A is the frontal area of the object
C a shape-dependent constant
ρ the air density



Initial Value Problem – still second order

$$m\frac{d^2x}{dt^2} = F_{(drag,x)}$$

$$m\frac{d^2y}{dt^2} = -mg + F_{(drag,y)}$$

$$F_{drag} = -(B_1v + B_2v^2)V$$

$$\frac{d^2x}{dt} = -\frac{B_1v_x}{m} - \frac{B_2v_x^2V}{m}$$

$$\frac{d^2y}{dt} = -g - \frac{B_1v_y}{m} - \frac{B_2Vv_y^2}{m}$$

Initial Value Problem – set of first order equation

The equations are second order; write them as a set of first order ODE:

$$\frac{dx}{dt} = v_x$$

$$\frac{dv_x}{dt} = -\frac{B_1 v_x}{m} - \frac{B_2 V v_x^2}{m}$$

$$\frac{dy}{dt} = v_y$$

$$\frac{dv_y}{dt} = -g - \frac{B_1 v_y}{m} - \frac{B_2 V v_y^2}{m}$$

Thus in vector form

$$\frac{dY}{dt} = f(Y)$$

$$Y = \begin{pmatrix} x \\ v_x \\ y \\ v_y \end{pmatrix}$$

$$f(Y) = \begin{pmatrix} v_x \\ -\frac{B_1 v_x}{m} - \frac{B_2 V v_x}{m} \\ v_y \\ -g - \frac{B_1 v_y}{m} - \frac{B_2 V v_y}{m} \end{pmatrix}$$

Euler Method

Simplest way to solve a set of ODE

$$\frac{dy}{dt} = f(t, y), \qquad y(t_0) = y_o$$

1 - discretize time using a step h

$$t_n = t_0 + nh, \text{ n=0, 1, } \ldots$$
$$y_n = y(t_n)$$

$$f'(x) = \frac{f(x+h) - f(x)}{h}$$
$$f(x+h) = f(x) + \text{h}f'(x)$$

2 - Approximate the derivative with a simple forward difference

$$y' = \text{D}_+(\text{h}) = \frac{y_{n+1} - y_n}{h} = f(t, y)$$

$$\boldsymbol{y_{n+1} = y_n + hf(t_n, y_n)}$$

3 - Calculate all y starting from y_0

$$y_1 = y_0 + hf(t_0, y_0)$$
$$y_2 = y_1 + hf(t_1, y_1) \ldots\ldots$$

Euler solution to kicking the ball

$$m\frac{d^2x}{dt^2} = F_{(drag,x)}$$

$$m\frac{d^2y}{dt^2} = -mg + F_{(drag,y)}$$

$$F_{drag} = -(B_1 v + B_2 v^2)V$$

The initial position (x, y) and initial velocity (v_x, v_y) are known at time t=0

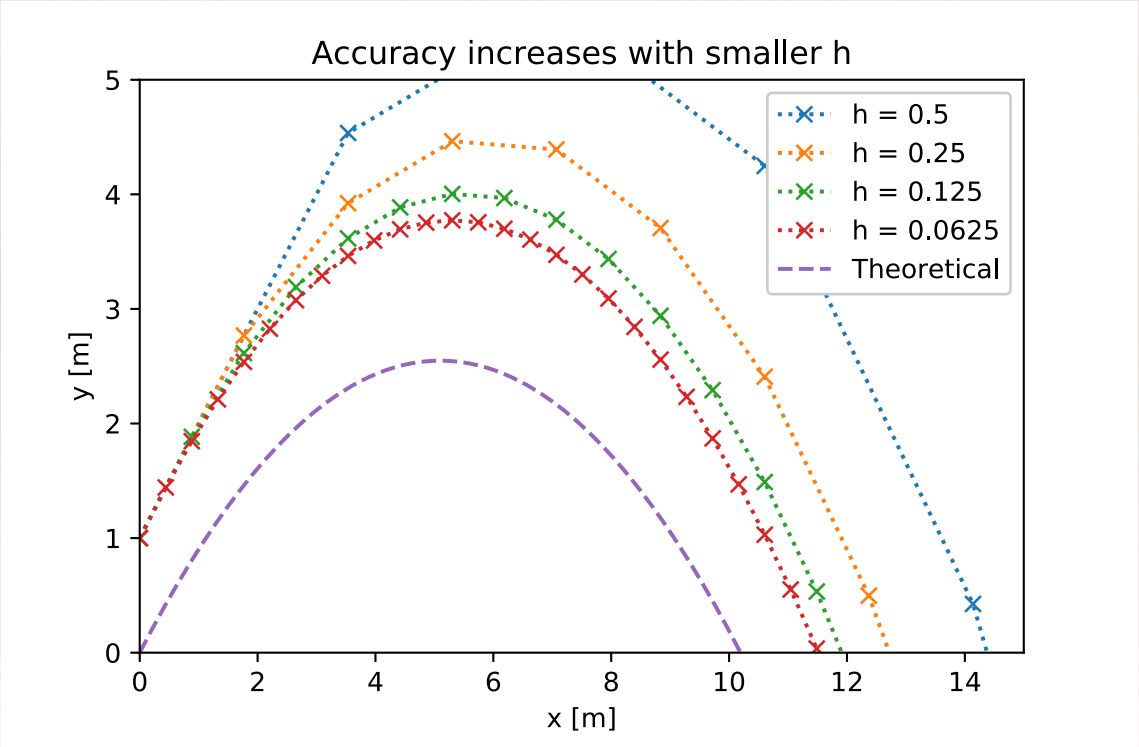
Solve the set of first order equations:

$$\frac{dx}{dt} = v_x$$

$$\frac{dv_x}{dt} = -\frac{B_1 v_x}{m} - \frac{B_2 V v_x}{m}$$

$$\frac{dy}{dt} = v_y$$

$$\frac{dv_y}{dt} = -g - \frac{B_1 v_y}{m} - \frac{B_2 V v_y}{m}$$



Estimating Errors

- **Data input Error**
- **Round-off Error**
- **Truncation Error**

Round-off Errors

The source - **Numerical calculations are done in integer or floating-point arithmetic.**
Floating-point arithmetic has round-off errors – it looks like:

$s \cdot M \cdot 2^p$

- s -sign
- M - Mantissa
- p – integer

Example: $(5.625)_{10} = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = (101.101)_2$ no Error

32 bits floating point – 1 bit for sign, 8 bits for p and 23 bits for mantissa

$$p = x^7 \cdot 2^7 + x^6 \cdot 2^6 + \ldots + x^0 \cdot 2^0 - 2^7 \qquad x_i \in \{0, 1\}$$

$$M = 1 + y_1 \cdot 2^{-1} + y_2 \cdot 2^{-2} + \ldots + y_{23} \cdot 2^{-23} \qquad y_i \in \{0, 1\}$$

Roundoff errors arise because the number of y_i's is finite.

Estimating Errors

Round-off Errors

x_r is the computer representation of x

Relative Error is $\frac{|x_r - x|}{|x|} = \mu = 2^{-t}$

For 32 bits $\mu = 2^{-23} \simeq 10^{-7}$
For 64 bits $\mu = 2^{-52} \simeq 10^{-16}$

Round-off errors make problems when subtracting numbers with small relative difference

Try and find tricks around it!

Estimating Errors

Round-off Errors

Trick Example

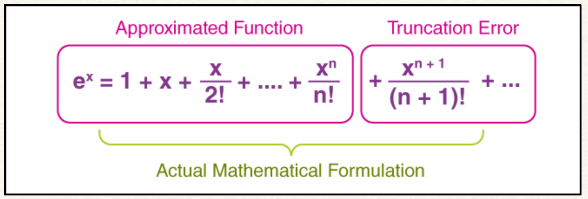
$ax^2+bx+c=0$ $\qquad x = -\frac{b - \sqrt{b^2 - 4ac}}{2a}$

If $b >> \sqrt{4ac}$ the numerator will be affected by round-off errors

The trick: multiply by $b + \sqrt{b^2 - 4ac}$ both nominator and denominator

$$x = -\frac{2c}{b + \sqrt{b^2 - 4ac}}$$ Not affected by round-off errors

Estimating Errors



Truncation Errors

Compute numerically the derivative f′(x)

$$f'(x) \approx D_+(h) = \frac{f(x+h) - f(x)}{h}$$

$$f'(x) \approx D_-(h) = \frac{f(x) - f(x-h)}{h}$$

$$f'(x) \approx D_0(h) = \frac{f(x+h) - f(x-h)}{2h}$$

Taylor expansion

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{3!}f'''(x) + \ldots.$$

move f(x) to the left
divide by h

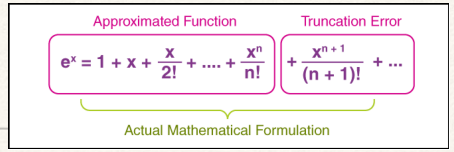
$$\frac{f(x+h)-f(x)}{h} = f'(x) + \frac{h}{2}f''(x) + \frac{h^2}{3!}f'''(x) + \ldots.$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{3!}f'''(x) + \ldots.$$

Thus the truncation error for $D_+(h)$ is : $\varepsilon_t = \frac{h}{2}f''(x) + \frac{h^2}{3}f'''(x) + \ldots.$

$$\epsilon_t \approx h|f''(x)|$$

Estimating Errors



Total relative Errors

For $D_+(h)$ Truncation Error $\epsilon_t \approx h|f''(x)|$

Relative Error is $\frac{|x_r - x|}{|x|} = \mu = 2^{-t}$

Round-off error for f(x+h) and f(x) is $\varepsilon_r \gtrsim \mu|f(x)|$ where μ is the floating point precision

For $D_+(h)$ the round-off error $\varepsilon_r \gtrsim \frac{\mu|f(x)|}{h}$

$$D_+(h) = \frac{f(x+h) - f(x)}{h}$$

If h is large ϵ_t is large, if h is small ε_r is large

Richardson Extrapolation

Reduces Truncation Error

$$\frac{f(x+h)-f(x)}{h} = f'(x) + \frac{h}{2}f''(x) + \frac{h^2}{3}f'''(x) + \ldots$$

Example: Assume we want to calculate $a_0 = \lim_{h \to o} a(h)$

1. Functional form $a(h)=a_0+a_1h^2+a_2h^4+\ldots$
2. The values of a(h) are known at h_0 $2h_0$, 2^2h_0

Improved estimator of a^0 is $\hat{a}(h)$

$$\left.\begin{array}{l} a(h)=a_0+a_1h^2+a_2h^4+O(h^6) \\ a(2h)=a_0+4a_1h^2+16a_2h^4+O(h^6) \end{array}\right\} \Rightarrow 4a(h) - a(2h) = 3a_0 - 12a_2h^4 + O(h^6)$$

$$\hat{a}(h) = \frac{4a(h) - a(2h)}{3} = a_0 - 4a_2h^4 + O(h^6)$$

$\hat{a}(h)$ is an improved estimator of a_0 because the truncation error is $O(h^4)$ instead of $O(h^2)$.

Further improved estimator

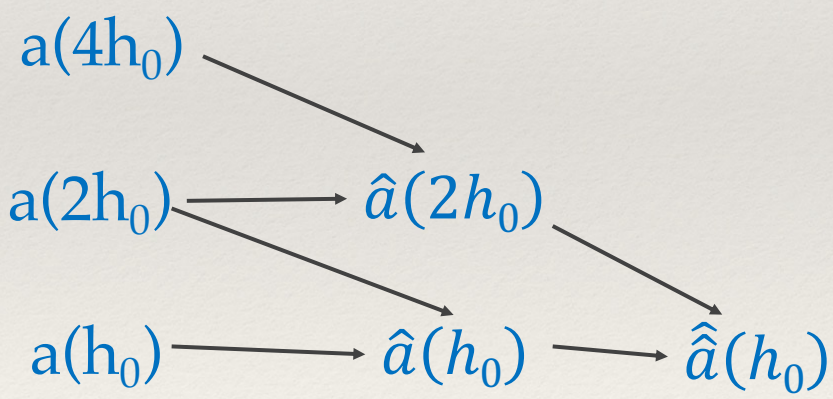
$$\hat{\hat{a}}(h) = \frac{16\hat{a}(h) - \hat{a}(2h)}{15} = a_0 + O(h^6)$$

Richardson Extrapolation

To calculate $\hat{\hat{a}}(h)$ for h = h_0 one needs the values of a(h) at h_0, $2h_0$ and $4h_0$

$$\hat{\hat{a}}(h) = \frac{16\hat{a}(\mathrm{h}) - \hat{a}(2\mathrm{h})}{15} \qquad \hat{a}(h) = \frac{4\mathrm{a}(\mathrm{h}) - \mathrm{a}(2\mathrm{h})}{3}$$

Therefore the Richardson Extrapolation scheme is as follows



Global vs. Local error

assume that the local error scales as h^m with step size, i.e., in step i we have a local error $C_i h^m$, where C_i is a constant.

$$E_L = C_i h^m$$

$$\frac{f(x+h)-f(x)}{h} = f'(x) + \frac{h}{2} f''(x) + \frac{h^2}{3} f'''(x) + \ldots$$

The global error is then the sum of the local errors

$$E_G = \sum_{i=1}^{N} C_i h^m$$

The upper bound for the global error is

$$E_G < \max(C_i) \sum_{i=1}^{N} h^m = \max(C_i)(\mathrm{Nh})\, h^{m-1}$$

Since t=Nh is kept constant the global error scales as h^{m-1} with step size h.

For Euler : the local truncation error i.e. the error at one step is $\sim h^2$,
the global truncation is $\sim h$

The Runge-Kutta Method

Euler Method - we calculate y_{n+1} using values of f at the begining of the interval - t_n and y_n

Runge-Kutta Methods - construct y_{n+1} using values of f at carefully chosen point between t_n and t_{n+1}

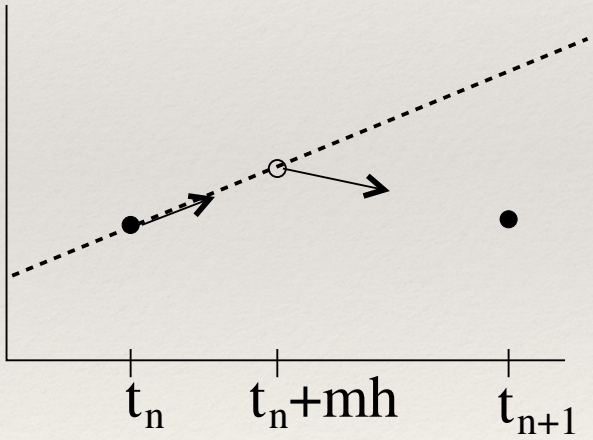
Second Order Runge-Kutta Method (local error $\sim h^3$)

Consider

$$k_1 = hf(t_n, y_n)$$

$$k_2 = hf(t_n + mh, y_n + mk_1)$$

$$y_{n+1} = y_n + ak_1 + bk_2$$



Second order RK – combines two values of f (k_1 and k_2) and has 3 parameters: a, b and m

Optimal parameters a+b=1 and bm =1/2

a=b=1/2 and m=1 Heuns' Method

The Runge-Kutta Method

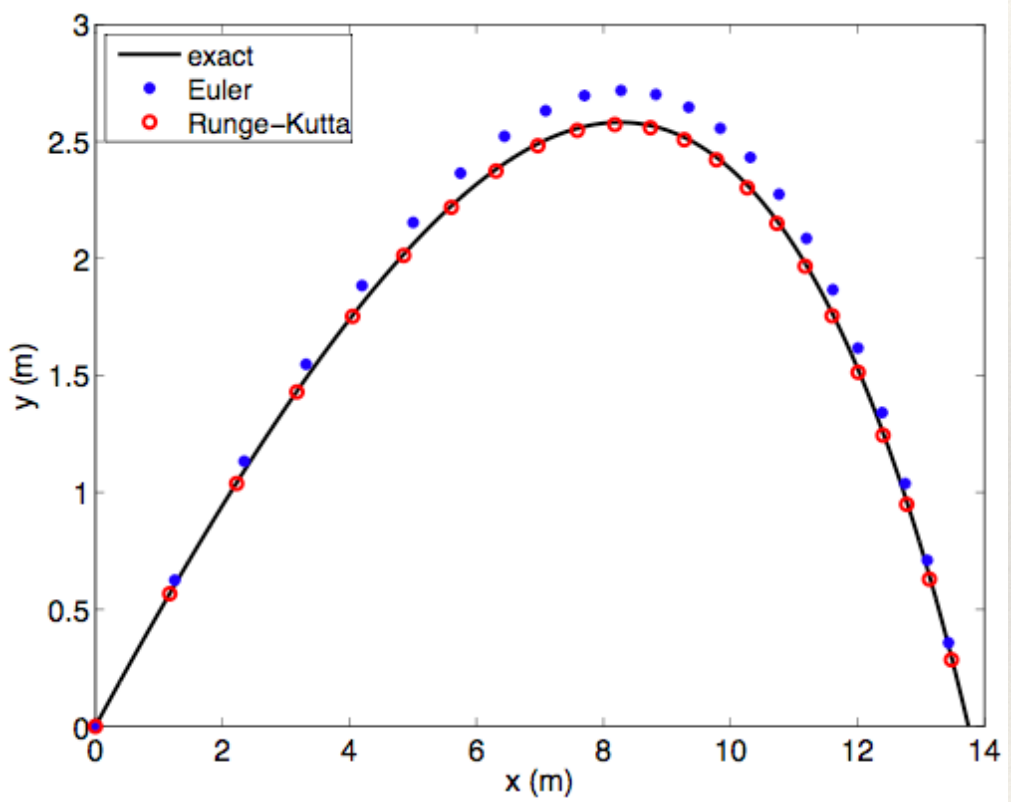
Comparison Euler vs. RK second order - kicking the ball example

$$\frac{dx}{dt} = v_x$$

$$\frac{dv_x}{dt} = -\frac{B_1 v_x}{m} - \frac{B_2 V v_x}{m}$$

$$\frac{dy}{dt} = v_y$$

$$\frac{dv_y}{dt} = -g - \frac{B_1 v_y}{m} - \frac{B_2 V v_y}{m}$$



Parameter values: h = 1/16 s, B2/m = 0.01m^{-1}, g = 9.82 m/s^2, v_x(t = 0) = 20 m/s and v_y(t = 0)=10 m/s. The ”exact” result is obtained using the Euler method with a very small step size.

The Runge-Kutta Method RK 4-5

Fourth Order Runge Kutta Method

adding more points gives improved accuracy at the cost of increased complexity.

$$k_1 = hf(t_n, y_n)$$

$$k_2 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$$

$$k_3 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right)$$

$$k_4 = hf(t_n + h, y_n + k_3)$$

$$y_{n+1} = y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(h^5)$$

Numerically Calculate Errors

Consider an $O(h^m)$ algorithm

When run with step 2h it produced output y_n^{2h} and global Error E_n^{2h}

$$Y_n^{exact} = y_n^{2h} + E_n^{2h} + \text{Higher Order Terms}$$

$$Y_n^{exact} = y_n^{2h} + C(2h)^m + \text{Higher Order Term}$$

We run the same algorithm with smaller step h

$$Y_n^{exact} = y_n^{h} + E_n^{h} + \text{Higher Order Terms}$$

$$Y_n^{exact} = y_n^{h} + C(h)^m + \text{Higher Order Term}$$

Substracting the terms above gives us the error:

$$E_n^h = \frac{y_n^h - y_n^{2h}}{2^m - 1}$$

$$0 = y_n^h + C(h)^m - y_n^{2h} - C(2h)^m$$

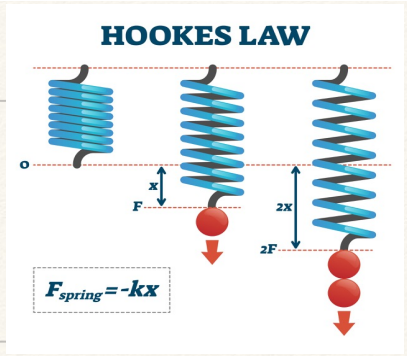
$$C(2h)^m - C(h)^m = y_n^h - y_n^{2h}$$

$$C(h)^m (2^m - 1) = y_n^h - y_n^{2h}$$

$$C(h)^m = \frac{y_n^h - y_n^{2h}}{2^m - 1}$$

It depends on m!

Instability



Assume $x_0=0$, Newton's equation of motion and Hooke's law

$$M\frac{d^2x}{dt^2} = -kx$$

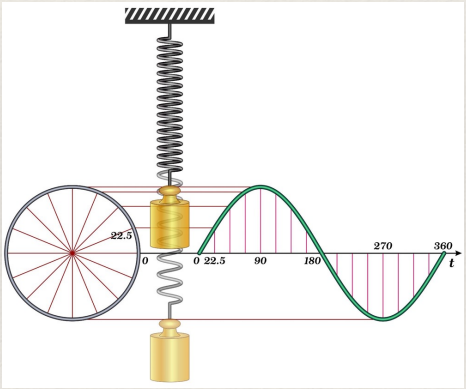
$$\frac{d^2x}{dt^2} = -(k/M)x$$

second order – has to be written in first order: $\frac{dy}{dt} = f(y)$

$$y = \begin{pmatrix} x \\ v_x \end{pmatrix}, f(y) = \begin{pmatrix} v_x \\ -\frac{k}{M}x \end{pmatrix}$$

Thus the system of equation is:

$$\begin{cases} \frac{dx}{dt} = v_x \\ \frac{dv_x}{dt} = -\frac{k}{M}x \end{cases}$$



Analitical solution :

If $v_o=0$ and A = x(0)

$$x(t) = A\cos\left(\sqrt{\frac{k}{M}}t\right)$$

Harmonic Oscillator

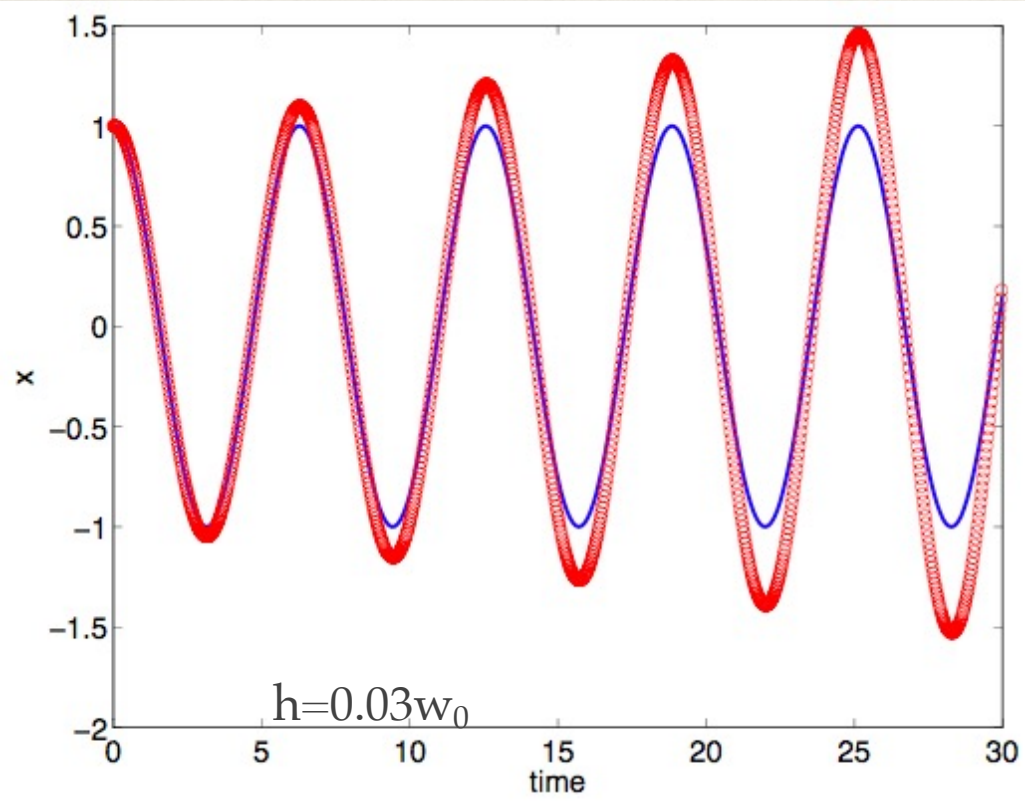
Instability

Numeric Euler

$t_n = t_0 + nh$, n=0, 1, …

$y_n = y(t_n)$

$y_{n+1} = y_n + hf(t_n, y_n)$



$$\begin{cases} \dfrac{dx}{dt} = v_x \\ \dfrac{dv_x}{dt} = -\dfrac{k}{M}x \end{cases}$$

Numerical

$$\begin{pmatrix} x_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} x_n \\ v_n \end{pmatrix} + h \begin{pmatrix} v_n \\ -\dfrac{k}{M}x_n \end{pmatrix}$$

Analitical

$$x(t) = A\cos\left(\sqrt{\frac{k}{M}}t\right)$$

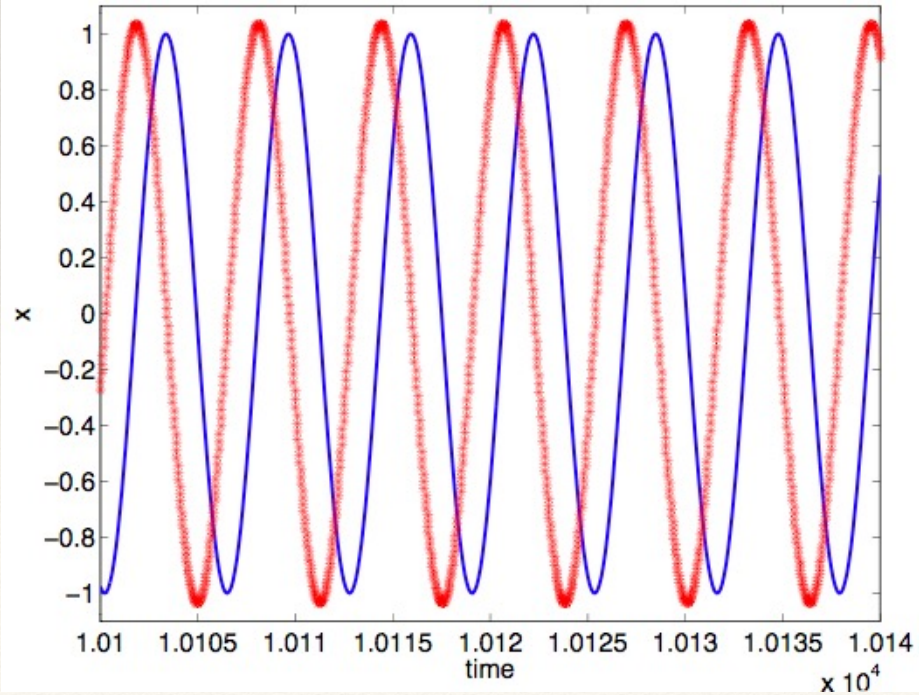
Numerical solution is unstable amplitude increases

Instability

Numeric Runge-Kutta

$$t_n = t_0 + nh,\ \text{n=0, 1, …}$$
$$y_n = y(t_n)$$
$$\left.\begin{aligned} k_1 &= hf(t_n, y_n) \\ k_2 &= hf(t_n + mh, y_n + mk_1) \end{aligned}\right\}$$
$$y_{n+1} = y_n + ak_1 + bk_2$$

h=0.03w_0



$$\begin{cases} \dfrac{dx}{dt} = v_x \\ \dfrac{dv_x}{dt} = -\dfrac{k}{M}\mathrm{x} \end{cases}$$

Numerical

$$k_1 = \begin{pmatrix} x_{temp1} \\ v_{temp1} \end{pmatrix} = h\begin{pmatrix} v_n \\ -\dfrac{k}{M}x_n \end{pmatrix}$$

$$k_2 = \begin{pmatrix} x_{temp2} \\ v_{temp2} \end{pmatrix} = h\begin{pmatrix} v_n + mv_{temp1} \\ -\dfrac{k}{M}(x_n + mx_{temp1}) \end{pmatrix}$$

$$\begin{pmatrix} x_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} x_n \\ v_n \end{pmatrix} + a\begin{pmatrix} x_{temp1} \\ v_{temp1} \end{pmatrix} + b\begin{pmatrix} x_{temp2} \\ v_{temp2} \end{pmatrix}$$

Analitical

$$x(t) = A\cos\left(\sqrt{\frac{k}{M}}t\right)$$

Numerical RK solution is also unstable

Instability - Explained

Numerical solutions are unstable – explained by the energy difference between steps

$$E = w_p + E_{kin} = \frac{kx^2}{2} + \frac{Mv^2}{2}$$

$$\begin{pmatrix} x_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} x_n \\ v_n \end{pmatrix} + h \begin{pmatrix} v_n \\ -\frac{k}{M} x_n \end{pmatrix}$$

Euler

$$E_{n+1} = \frac{kx_{n+1}^2}{2} + \frac{Mv_{n+1}^2}{2} = \frac{k^2(x_n + hv_n)}{2} + \frac{M}{2}\left(v_n - h\frac{k}{M}x_n\right)^2$$

$$= \frac{kx_n^2}{2} + \frac{Mv_n^2}{2} + \frac{k}{2}(x_n h v_n - v_n h x_n) + \left(\frac{k}{2}{v_n}^2 + \frac{k^2}{2M}{x_n}^2\right)h^2 \Rightarrow$$

$$= E_n + 0 + Ch^2$$

$$E_{n+1} = E_n + Ch^2$$

Energy increases for each time step!!

The Euler-Cromer Method

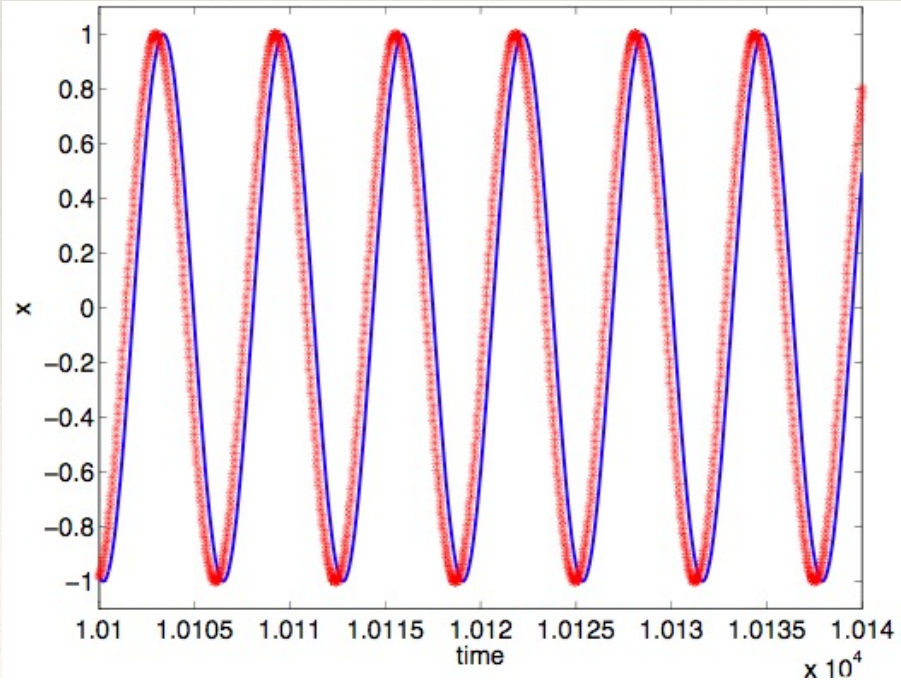
Cromer suggested a simple solution to the stability problem. The idea to use Euler's method, but when updating the position at step n + 1 one uses v_{n+1} instead of v_n.

Euler

$$\begin{pmatrix} x_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} x_n \\ v_n \end{pmatrix} + h \begin{pmatrix} v_n \\ -\frac{k}{M} x_n \end{pmatrix}$$

Euler Cromer

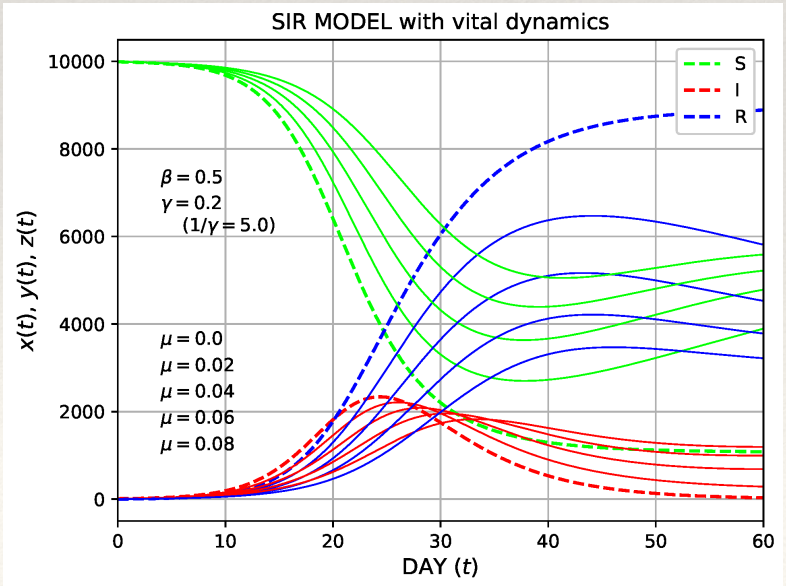
$$\begin{pmatrix} x_{n+1} \\ v_{n+1} \end{pmatrix} = \begin{pmatrix} x_n \\ v_n \end{pmatrix} + h \begin{pmatrix} \boldsymbol{v_{n+1}} \\ -\frac{k}{M} x_n \end{pmatrix}$$



No Instability

Projects

Disease spreading



Cell Reprogramming