Programovanie, 2. prednáška

Peter Vanya
Jarná škola FX

14. apríla 2014

Derivácie

Z teórie vieme, že

$$\frac{dy}{dx} = \lim_{h \to 0} \frac{y(x+h) - y(x)}{h}$$

Z toho plynie Newtonov vzťah

$$f(x+h) \approx f(x) + f'(x)h$$

V počítači volíme konkrétnu veľkosť h, ktorá by ale mala byť dostatočne malá. Keď máme body x_i a $y(x_i)$, a $x_{i+1} - x_i = h$, $\forall i$ tak ich diskrétna derivácia je

$$\frac{dy}{dx} = \frac{y(x_{i+1}) - y(x_i)}{h} + \mathcal{O}(h)$$

Derivácie

Vieme ale byť presnejší za použitia tých istých diskrétnych bodov a trochy rozumu. Deriváciu možno odvodiť z Taylorovho rozvoja:

$$f(x+h) = f(x) + f'(x)h + f''(x)\frac{h^2}{2} + f'''(x)\frac{h^3}{6} + \mathcal{O}(h^4)$$

$$f(x-h) = f(x) - f'(x)h + f''(x)\frac{h^2}{2} - f'''(x)\frac{h^3}{6} + \mathcal{O}(h^4)$$

Odčítaním dostávame (cvičenie)

$$\frac{dy}{dx} = \frac{y(x+h) - y(x-h)}{2h} + \mathcal{O}(h^2)$$

Diferenciálne rovnice

DR prvého rádu:

$$\frac{dy(x)}{dx} = y'(x) = f(x, y)$$

Zoberme si niečo neriešiteľné:

$$y' = \sqrt{xy + 1}$$

čo s tým? Poznáme x_0 a y_0, takže $y'_0 = \sqrt{x_0 y_0 + 1}$

$$y_1 = y_0 + h\, y'_0$$
$$x_1 = x_0 + h$$

That's it! A tak dookola, od y_1 až po y_N. Tomuto postupu sa hovorí **Eulerova metóda**.

Runge-Kuttova metóda

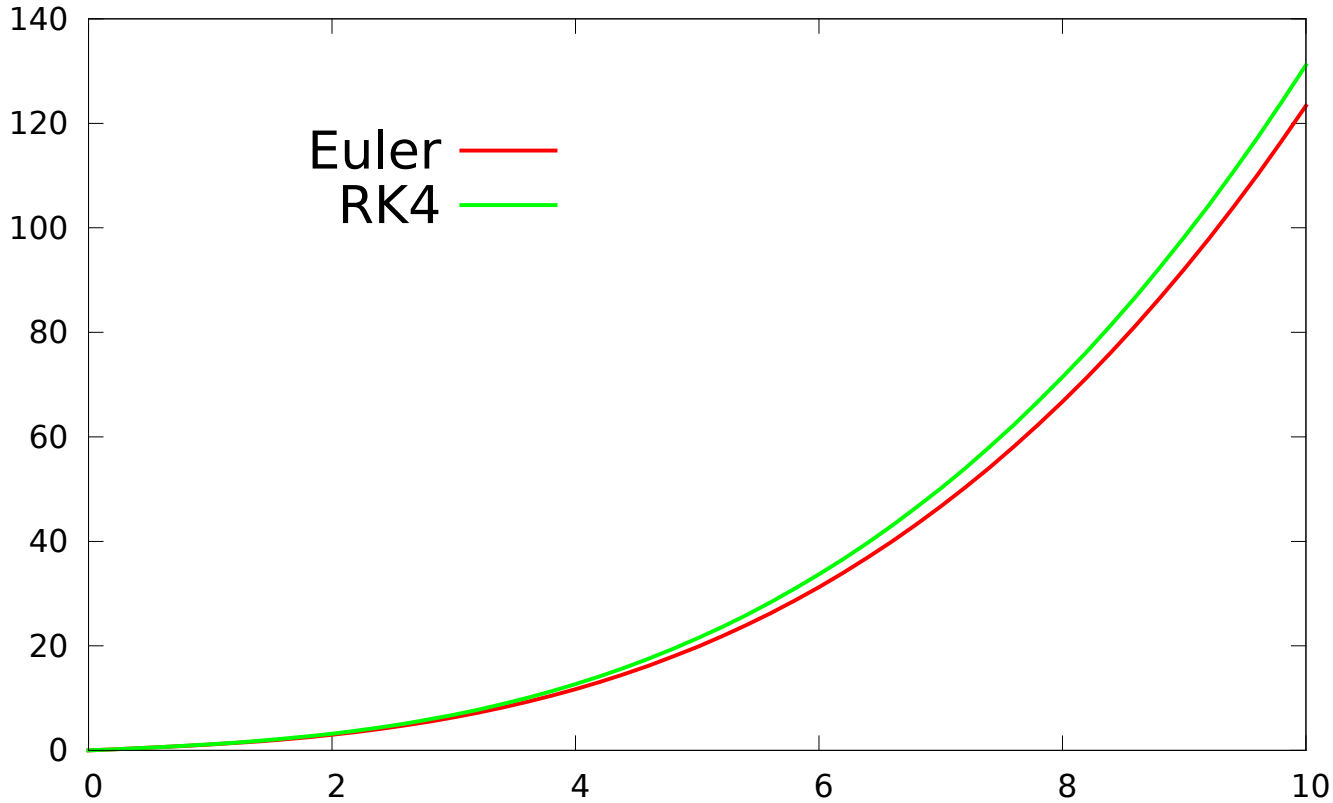
Eulerova metóda ale nebýva veľmi presná, preto sa používa napr. **leapfrog** alebo hlavne **Runge-Kutta 4**. Pre jeden x-ový skok h treba urobiť štyri medzivýpočty:

$$\begin{aligned} k_1 &= f(x_i, y_i) \\ k_2 &= f(x_i + \frac{1}{2}h, y_i + \frac{h}{2}k_1) \\ k_3 &= f(x_i + \frac{1}{2}h, y_i + \frac{h}{2}k_2) \\ k_4 &= f(x_i + h, y_i + hk_3) \end{aligned}$$

Teraz to „zlepíme dokopy“:

$$\begin{aligned} y_{i+1} &= y_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ x_{i+1} &= x_i + h \end{aligned}$$

Euler vs RK4



Strela v odporovom prostredí

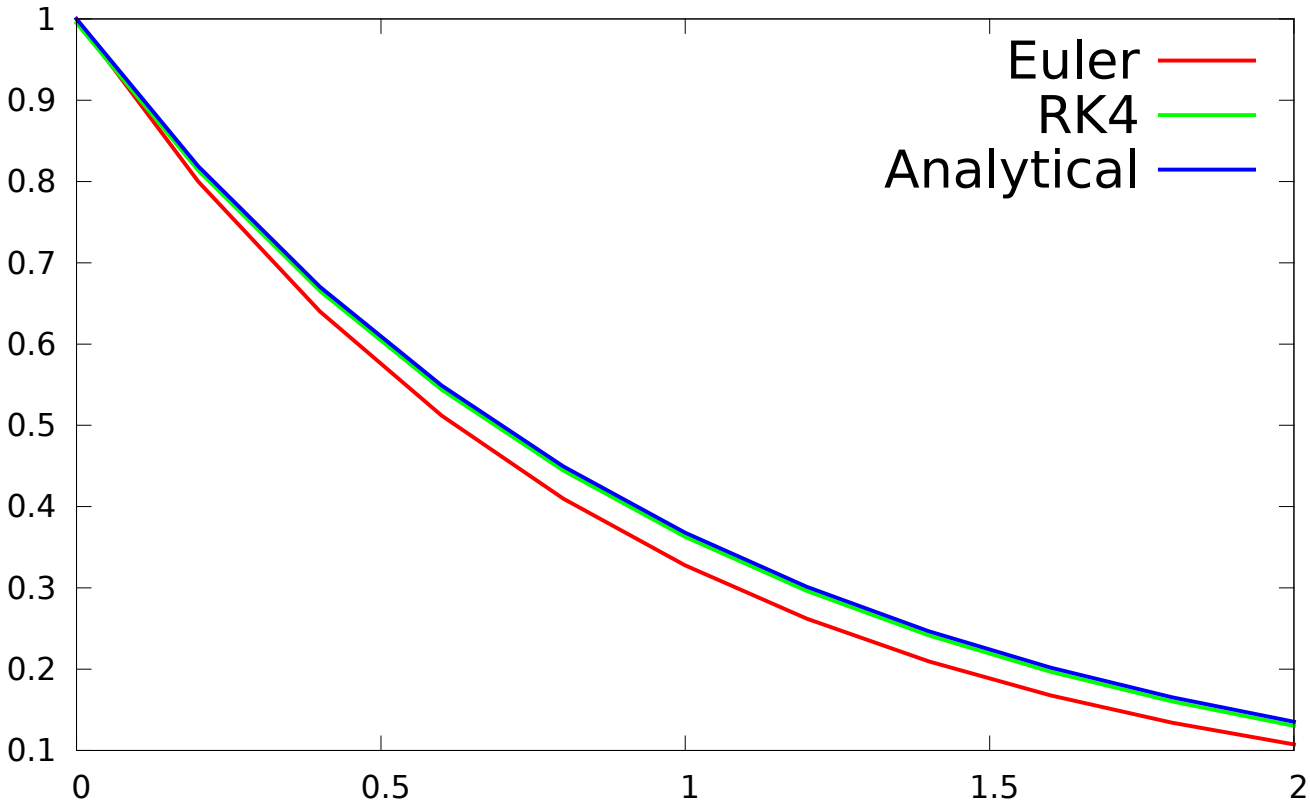
Napr. vo vode bez turbulencií (Stokesov vzťah $F_o = 6\pi\eta\rho v$) je popísaná diferenciálnou rovnicou

$$ma = -kv$$

V jazyku počítača Eulerovou metódou to znamená:

$$a_i = -\frac{k}{m} v_i$$
$$v_{i+1} = v_i + a_i \Delta t$$
$$x_{i+1} = x_i + v_i \Delta t$$

Graf



Vidíme, že tu sa Eulerova metóda začína rozchádzať s ideálom.

RK4

Pre DR 2. rádu je táto metóda o trochu ťažšia.
Všimnime si maticovú štruktúru predošlých rovníc:

$$\begin{pmatrix} v_{i+1} \\ x_{i+1} \end{pmatrix} = \begin{pmatrix} v_i \\ x_i \end{pmatrix} + \begin{pmatrix} -\frac{k}{m} & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} v_i \\ x_i \end{pmatrix} \Delta t$$

Teda

$$\mathbf{a}_{i+1} = \mathbf{a}_i + A\mathbf{a}_i \Delta t = \mathbf{a}_i + f(\mathbf{a}_i)\Delta t$$

Toto sa nápadne podobá na Eulerovu metódu, kde $\mathbf{a}' = f(\mathbf{a})$ je matica A.

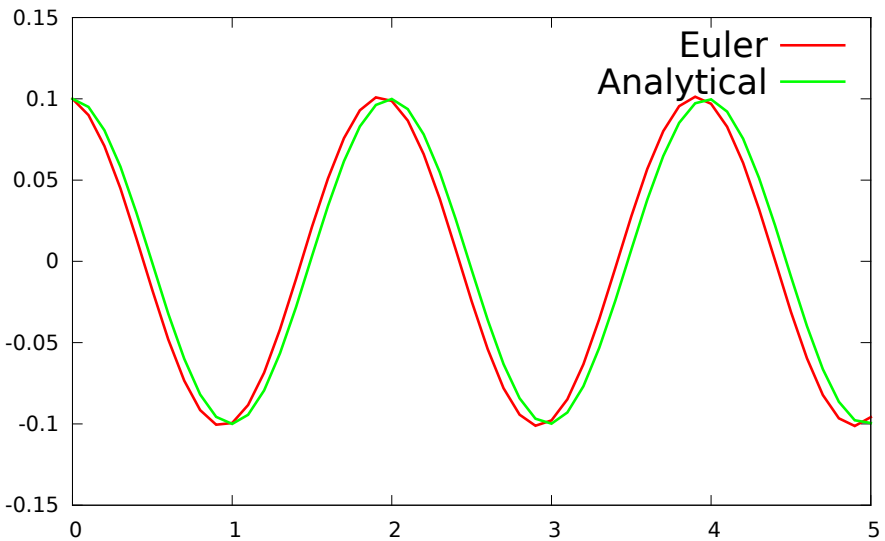
Podľa tejto logiky sú koeficienty k_i v RK4 metóde vektormi, ktoré vznikajú násobením matice A.

Matematické kyvadlo

$$\ddot{\theta} = -\frac{g}{l}\theta$$

Matica A tak je (cvičenie)

$$A = \begin{pmatrix} 0 & -\frac{g}{l} \\ 1 & 0 \end{pmatrix}$$



Vektory

Ako reprezentovať matice? C++ sa veľmi s maticami nekamaráti, definujú sa takým obskúrnym spôsobom. Najskôr sa pozrime ako možno alternatívne zadefinovať vektor.

```
int N = 3;
double *a = new double[N];
for(int i=0; i<N; i++)
  a[i] = i;     // rovnaky pristup ako pri poli
delete[] a;
```

Riadok `new double[N]` *alokuje N* miest v pamäti o veľkosti `double`. Hviezdička značí tzv **pointer**, ktorý ukazuje na tých *N* miest. Výhodou je, že takéto pole môže byť vracané funkciou, pretože pointer je de facto len jedna premenná.
Pred skončením programu sa patrí tieto miesta zrušiť (*dealokovať*) príkazom `delete[]`, aby nám zbytočne nezaberali pamäť.

Matice

Podobne definujeme maticu – je to pointer na pole pointrov.

```
int N = 3;
double **A = new double *[N];
for(int i=0; i<N; i++)
  A[i] = new double[N];
...
for(int i=0; i<N; i++)
  delete[] A[i];
delete[] A;
```

K prvkom matice pristupujeme veľmi intuitívne, cez `A[i][j]`.

Teraz si napr. môžeme napísať funkciu na násobenie matíc, resp. vektora a matice, ktorá na výstupe bude dávať maticu, resp. vektor.