Komputerowa analiza zagadnień różniczkowych

9. Równania drugiego rzędu

P. F. Góra

http://th-www.if.uj.edu.pl/zfs/gora/

2013

Motywacja

Ponieważ klasyczne równania ruchu (równania Newtona) mają postać równań różniczkowych drugiego rzędu, w praktyce bardzo ważna jest umiejętność numerycznego rozwiązywania równań postaci $m_i\ddot{\mathbf{x}}_i = \mathbf{F}_i$, $i = 1, 2, \ldots, N$, czyli

$$\frac{d^2\mathbf{x}}{dt^2} = \mathbf{a}(t, \mathbf{x}(t)) \,, \tag{1}$$

gdzie $\mathbf{x}, \mathbf{a} \in \mathbb{R}^{3N}$, N jest ilością cząstek*. Równanie tej postaci można łatwo przekształcić do układu równań pierwszego rzędu, ale szczególnie efektywne powinny być metody „od razu” dostosowane do równań rzędu drugiego.

*Niekiedy ruch jest z przyczyn fizycznych ograniczony do przypadku dwu- lub jednowymiarowego. Wówczas zamiast $3N$ w (1) mamy odpowiednio $2N$ lub N.

Przykład — metoda punktu środkowego

Równanie (1) możemy zamienić na układ równań pierwszego rzędu

$$\frac{d\mathbf{x}}{dt} = \mathbf{v}\,, \tag{2a}$$

$$\frac{d\mathbf{v}}{dt} = \mathbf{a}(t,\mathbf{x})\,. \tag{2b}$$

Formalnie możemy ten układ zapisać jako $\frac{d\vec{\mathbf{x}}}{dt} = \vec{\mathbf{f}}(t,\vec{\mathbf{x}})$, gdzie $\vec{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ \mathbf{v} \end{bmatrix}$.

Do układu (2) zastosujmy jawną metodę punktu środkowego.

Otrzymujemy

$$\vec{\mathbf{k}}_1 = \begin{bmatrix} \mathbf{v}_n \\ \mathbf{a}(t_n, \mathbf{x}_n) \end{bmatrix}, \qquad \vec{\mathbf{x}}_n + \frac{1}{2}h\vec{\mathbf{k}}_1 = \begin{bmatrix} \mathbf{x}_n + \frac{1}{2}h\mathbf{v}_n \\ \mathbf{v}_n + \frac{1}{2}h\mathbf{a}(t_n, \mathbf{x}_n) \end{bmatrix} \tag{3a}$$

$$\vec{\mathbf{k}}_2 = \begin{bmatrix} \mathbf{v}_n + \frac{1}{2}h\mathbf{a}(t_n, \mathbf{x}_n) \\ \mathbf{a}\left(t_n + \frac{1}{2}h, \mathbf{x}_n + \frac{1}{2}h\mathbf{v}_n\right) \end{bmatrix} \tag{3b}$$

Ostatecznie

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathbf{v}_n + \frac{1}{2}h^2\mathbf{a}(t_n, \mathbf{x}_n) \tag{4a}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + h\mathbf{a}\left(t_n + \frac{1}{2}h, \mathbf{x}_n + \frac{1}{2}h\mathbf{v}_n\right) \tag{4b}$$

Zwróćmy uwagę, na charakterystyczną obecność wielkości h^2 w wyrażeniu na $\mathbf{x}_{n+1}$.

Dygresja

Porównajmy algorytm (4) z **bardzo złym** algorytmem

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathbf{v}_n + \frac{1}{2}h^2\mathbf{a}(t_n, \mathbf{x}_n) \tag{5a}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + h\mathbf{a}\left(t_n, \mathbf{x}_n\right). \tag{5b}$$

Powyższy algorytm powstał w wyniku naiwnego zastosowania "równań na ruch jednostajnie przyspieszony" do problemu (1). Algorytm ten daje niewłaściwe wyniki. Niestety, można go znaleźć w niektórych podręcznikach ☹

Metody Rungego-Kutty-Nyströma

Podejście, jakiego użyliśmy do wyprowadzenia algorytmu (4), można uogólnić. Otrzymujemy w ten sposób metody Rungego-Kutty-Nyströma (RKN) dla równań postaci (1):

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathbf{x}'_n + h^2 \sum_{i=1}^{s} w_i \mathbf{k}_i \tag{6a}$$

$$\mathbf{x}'_{n+1} = \mathbf{x}'_n + h \sum_{i=1}^{s} \bar{w}_i \mathbf{k}_i \tag{6b}$$

$$\mathbf{k}_i = \mathbf{a}\left(t_n + \alpha_i h, \mathbf{x}_n + \alpha_i h \mathbf{x}'_n + h^2 \sum_{j=1}^{s} \beta_{ij} \mathbf{k}_j \right) \tag{6c}$$

Zestawy wag $\{w_i\}$, $\{\bar{w}_i\}$ mogą być różne. Istnieje wiele różnych metod RKN, w tym metody niejawne, zagnieżdżone.

Przykład metody RKN rzędu czwartego

Rozważamy równanie postaci $\frac{d^2\mathbf{x}}{dt^2} = \mathbf{a}(t, \mathbf{x}, \mathbf{x}')$ (siły mogą zależeć od prędkości). Metodą RKN jest[†]

$$\mathbf{k}_1 = \mathbf{a}\left(t_n, \mathbf{x}_n, \mathbf{x}'_n\right) \tag{7a}$$

$$\mathbf{k}_2 = \mathbf{a}\left(t_n + \tfrac{1}{2}h, \mathbf{x}_n + \tfrac{1}{2}h\mathbf{x}'_n + \tfrac{1}{8}h^2\mathbf{k}_1, \mathbf{x}'_n + \tfrac{1}{2}h\mathbf{k}_1\right) \tag{7b}$$

$$\mathbf{k}_3 = \mathbf{a}\left(t_n + \tfrac{1}{2}h, \mathbf{x}_n + \tfrac{1}{2}h\mathbf{x}'_n + \tfrac{1}{8}h^2\mathbf{k}_2, \mathbf{x}'_n + \tfrac{1}{2}h\mathbf{k}_2\right) \tag{7c}$$

$$\mathbf{k}_4 = \mathbf{a}\left(t_n + h, \mathbf{x}_n + h\mathbf{x}'_n + h^2\mathbf{k}_3, \mathbf{x}'_n + h\mathbf{k}_3\right) \tag{7d}$$

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathbf{x}'_n + \tfrac{1}{6}h^2(\mathbf{k}_1 + \mathbf{k}_2 + \mathbf{k}_3) \tag{7e}$$

$$\mathbf{x}'_{n+1} = \mathbf{x}'_n + \tfrac{1}{6}h(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4) \tag{7f}$$

[†]Sprawdzić to!

Algorytmy symplektyczne

Rozważać teraz będziemy układy hamiltonowskie, to jest układy $\mathbf{q}, \mathbf{p} \in \mathbb{R}^n$, przy czym zakładamy, że istnieje funkcjonał, zwany hamiltonianem,

$$H(\mathbf{q}, \mathbf{p}) : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R} \tag{8}$$

taki, że równania ruchu mają postać

$$\frac{d\mathbf{q}}{dt} = \nabla_{\mathbf{p}} H(\mathbf{q}, \mathbf{p}) \,, \qquad \frac{d\mathbf{p}}{dt} = -\nabla_{\mathbf{q}} H(\mathbf{q}, \mathbf{p}) \,. \tag{9}$$

(Po rozpisaniu (9) na składowe, otrzymujemy $\dot{q}_i = \partial H/\partial p_i$, $\dot{p}_i = -\partial H/\partial q_i$.)

Wszystkie układy mechaniki klasycznej bez dysypacji są hamiltonowskie.

Jeżeli zdefiniujemy $\mathbf{z} = \{\mathbf{q}, \mathbf{p}\}$, równania (9) możemy zapisać w postaci

$$\frac{d\mathbf{z}}{dt} = \mathbf{J}\,\nabla_{\mathbf{z}} H(\mathbf{z})\,, \tag{10}$$

gdzie

$$\mathbf{J} = \begin{bmatrix} 0 & \mathbb{I} \\ -\mathbb{I} & 0 \end{bmatrix} \tag{11}$$

nazywa się *macierzą symplektyczną*.

Twierdzenie

Niech $\mathbf{z}_1(0) = \{\mathbf{q}_1(0), \mathbf{p}_1(0)\}$, $\mathbf{z}_2(0) = \{\mathbf{q}_2(0), \mathbf{p}_2(0)\}$ będą dwoma *dowolnymi* warunkami początkowymi, natomiast $\mathbf{z}_1(t) = \{\mathbf{q}_1(t), \mathbf{p}_1(t)\}$, $\mathbf{z}_2(t) = \{\mathbf{q}_2(t), \mathbf{p}_2(t)\}$ niech będą odpowiadającymi im rozwiązaniami równania (10). Wówczas

$$\mathbf{z}_1^T(t)\,\mathbf{J}\,\mathbf{z}_2(t) = \mathbf{z}_1^T(0)\,\mathbf{J}\,\mathbf{z}_2(0)\,. \tag{12}$$

Algorytm numerycznego rozwiązywania hamiltonowskich ODE, który spełnia (12), nazywa się *algorytmem symplektycznym*.

Ważna obserwacja

Jeśli równania ruchu układu hamiltonowskiego rozwiązujemy numerycznie algorytmem symplektycznym, całki ruchu *nie* są, ściśle rzecz biorąc, zachowane, ale nie oddalają się bardzo od swoich *dokładnych* wartości, lekko wokół nich oscylując.

Symplektyczne metody RK

Rozważmy s-krokową metodę Rungego-Kutty opisaną tabelką

$$\begin{array}{c|cccc} \alpha_1 & \beta_{11} & \beta_{12} & \dots & \beta_{1s} \\ \alpha_2 & \beta_{21} & \beta_{22} & \dots & \beta_{2s} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \alpha_s & \beta_{s1} & \beta_{s2} & \dots & \beta_{ss} \\ \hline & w_1 & w_2 & \dots & w_s \end{array} \tag{13}$$

Niech $\mathbf{M} \in \mathbb{R}^{s \times s}$ będzie macierzą o elementach

$$\mathbf{M}_{ij} = w_i \beta_{ij} + w_j \beta_{ji} - w_i w_j \, . \tag{14}$$

Twierdzenie: Jeżeli $\mathbf{M} = 0$, metoda (13) jest symplektyczna. (Jest to także warunek konieczny.)

Przykład

Dwukrokowa metoda Gaussa-Legendre'a

$$\begin{array}{c|cc} \frac{1}{2}-\frac{\sqrt{3}}{6} & \frac{1}{4} & \frac{1}{4}-\frac{\sqrt{3}}{6} \\ \frac{1}{2}+\frac{\sqrt{3}}{6} & \frac{1}{4}+\frac{\sqrt{3}}{6} & \frac{1}{4} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \tag{15}$$

jest symplektyczna.

Obserwacja: Tylko *niejawne* metody RK mogą być symplektyczne

Dlaczego? Zgodnie z równaniem (14),

$$\forall i : \mathbf{M}_{ii} = 2w_i\beta_{ii} - w_i^2 = 0 \rightsquigarrow \beta_{ii} \neq 0\,, \tag{16}$$

o ile tylko $w_i \neq 0$. Ale *wszystkie* w_i nie mogą jednocześnie znikać.

Problem: Niejawne metody Rungego-Kutty są kosztowne w użyciu ☹

Hamiltoniany separowalne

Załóżmy, że hamiltonian badanego układu jest *separowalny*, to znaczy ma postać

$$H(\mathbf{p},\mathbf{q}) = T(\mathbf{p}) + U(\mathbf{q}) \tag{17}$$

Bardzo wiele "ważnych" hamiltonianów ma taką postać. T jest energią kinetyczną, U — energią potencjalną. W takim wypadku równania ruchu mają postać

$$\dot{\mathbf{q}} = \mathbf{P}(\mathbf{p}) = \nabla_{\mathbf{p}} T(\mathbf{p})\,, \qquad \dot{\mathbf{p}} = \mathbf{F}(\mathbf{q}) = -\nabla_{\mathbf{q}} U(\mathbf{q})\,. \tag{18}$$

Siła to minus gradient potencjału ☺

Symplektyczne algorytmy wyższych rzędów — algorytm Candy-Rozmusa

Przypuśćmy, iż mamy separowalny hamiltonian (17); równania ruchu mają wobec tego postać (18). Krok całkowania oznaczam przez h. W chwili t_n układ jest w punkcie $(\mathbf{p}_n, \mathbf{q}_n)$. Przyjmijmy $\tilde{\mathbf{q}}_0 = \mathbf{q}_n$, $\tilde{\mathbf{p}}_0 = \mathbf{p}_n$. Algorytm: Dla $i = 1, \ldots, 4$:

$$\tilde{\mathbf{p}}_i = \tilde{\mathbf{p}}_{i-1} + h\, b_i\, \mathbf{F}(\tilde{\mathbf{q}}_{i-1})\,, \quad \tilde{\mathbf{q}}_i = \tilde{\mathbf{q}}_{i-1} + h\, a_i\, \mathbf{P}(\tilde{\mathbf{p}}_i)\,, \tag{19a}$$

gdzie

$$a_1 = a_4 = (2 + 2^{1/3} + 2^{-1/3})/6\,, \tag{19b}$$

$$a_2 = a_3 = (1 - 2^{1/3} - 2^{-1/3})/6\,, \tag{19c}$$

$$b_1 = 0\,, \tag{19d}$$

$$b_2 = b_4 = 1/(2 - 2^{1/3})\,, \tag{19e}$$

$$b_3 = 1/(1 - 2^{1/3})\,. \tag{19f}$$

Bierzemy $\mathbf{p}_{n+1} = \tilde{\mathbf{p}}_4 + O(h^5)$, $\mathbf{q}_{n+1} = \tilde{\mathbf{q}}_4 + O(h^5)$.

Algorytm Ruth

Algorytm Candy-Rozmusa jest czwartego rzędu. *Algorytm Ruth* jest trzeciego rzędu, ma taką samą strukturę, jak Candy-Rozmus, ale ma współczynniki

$$(a_1, a_2, a_3) = \left(\frac{2}{3}, -\frac{2}{3}, 1\right), \tag{20a}$$

$$(b_1, b_2, b_3) = \left(\frac{7}{24}, \frac{3}{4}, -\frac{1}{24}\right). \tag{20b}$$

Algorytmy Candy-Rozmusa i Ruth *nie* są algorytmami Rungego-Kutty (różne współczynniki dla $\mathbf{p}$, $\mathbf{q}$), ale są "w stylu" Rungego-Kutty.

Metody Verleta — motywacja

Algorytmy RKN i algorytmy symplektyczne są kosztowne w użyciu — wymagają obliczeń w punktach pośrednich, a więc kilku obliczeń przyspieszenia (siły) w każdym kroku całkowania. Jeżeli badamy układ *wielu* oddziałujących cząstek, to, po pierwsze, każde obliczenie siły może być bardzo kosztowne, po drugie, tak naprawdę nie interesuje nas trajektoria układu, ale pewne własności statystyczne, dotyczące całego zespołu badanych cząstek. Dlatego w tego typu sytuacjach wygodnie jest mieć algorytm mniej dokładny, ale za to szybki w użyciu.

Klasyczny algorytm Verleta

Spróbujmy wyprowadzić metodę całkowania równania (1) w sposób systematyczny. Zakładamy, że przyspieszenia nie zależą od prędkości. Dokonujemy następujących rozwinięć w szereg Taylora:

$$\mathbf{x}(t_n+h) = \mathbf{x}(t_n) + \left.\frac{d\mathbf{x}}{dt}\right|_{t_n} h + \frac{1}{2}\left.\frac{d^2\mathbf{x}}{dt^2}\right|_{t_n} h^2 + \frac{1}{6}\left.\frac{d^3\mathbf{x}}{dt^3}\right|_{t_n} h^3 + O(h^4) \qquad (21a)$$

$$\mathbf{x}(t_n-h) = \mathbf{x}(t_n) - \left.\frac{d\mathbf{x}}{dt}\right|_{t_n} h + \frac{1}{2}\left.\frac{d^2\mathbf{x}}{dt^2}\right|_{t_n} h^2 - \frac{1}{6}\left.\frac{d^3\mathbf{x}}{dt^3}\right|_{t_n} h^3 + O(h^4) \qquad (21b)$$

Po dodaniu równań (21) stronami i uporządkowaniu wyrazów otrzymujemy

$$\mathbf{x}_{n+1} = 2\mathbf{x}_n - \mathbf{x}_{n-1} + \mathbf{a}_n h^2 + O(h^4)\,. \qquad (22)$$

Algorytm (22) zwany jest *metodą (algorytmem) Verleta* (Verlet 1967).

Zauważmy, iż metoda Verleta jest *metodą wielokrokową* — do obliczenia przyszłej wartości $\mathbf{x}$ potrzebna jest znajomość obecnej *oraz* poprzedniej wartości $\mathbf{x}$.

W celu pokazania, iż algorytm Verleta jest zgodny z równaniem (1), przekształcamy równanie (22) do postaci

$$\frac{\frac{\mathbf{x}_{n+1}-\mathbf{x}_n}{h} - \frac{\mathbf{x}_n-\mathbf{x}_{n-1}}{h}}{h} = \mathbf{a}_n \,. \tag{23}$$

W granicy $h \to 0^+$ odtwarzamy równanie (1).

Stabilność metody Verleta

Z uwagi na to, że jest to metoda wielokrokowa, stabilność algorytmu Verleta bada się w sposób właściwy dla tej klasy algorytmów. Dla uproszczenia notacji przyjmijmy na chwilę, iż interesuje nas wyłącznie przypadek jednowymiarowy. Mamy zatem

$$\begin{aligned} x_{n+1} + \varepsilon_{n+1} &= 2(x_n + \varepsilon_n) - (x_{n-1} + \varepsilon_{n-1}) + h^2 a(x_n + \varepsilon_n) \\ &\simeq 2x_n - x_{n-1} + h^2 \underbrace{a(x_n)}_{=a_n} + 2\varepsilon_n - \varepsilon_{n-1} + h^2 \left.\frac{da}{dx}\right|_{t_n} \varepsilon_n \end{aligned} \tag{24}$$

Nawias w napisie $a(x_n + \varepsilon_n)$ nie oznacza mnożenia, ale zależność funkcyjną!

Propagacja błędu opisana jest zatem równaniem

$$\varepsilon_{n+1} = (2+\lambda)\varepsilon_n - \varepsilon_{n-1}\,, \tag{25}$$

gdzie $\lambda = da/dx|_{t_n} h^2$. Równanie (25) możemy przedstawić w postaci[‡]

$$\begin{bmatrix} \varepsilon_{n+1} \\ \varepsilon_n \end{bmatrix} = \mathbf{G} \begin{bmatrix} \varepsilon_n \\ \varepsilon_{n-1} \end{bmatrix} = \begin{bmatrix} 2+\lambda & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \varepsilon_n \\ \varepsilon_{n-1} \end{bmatrix}. \tag{26}$$

Równaniem charakterystycznym jest

$$\det(\mathbf{G} - g\mathbb{I}) = -g(2+\lambda-g)+1 = g^2 - (2+\lambda) + 1 = 0 \tag{27}$$

[‡]Wielokrokowa propagacja błędu jest jednokrokowa w przestrzeni o odpowiednio większej ilości wymiarów!

Widać zatem, iż algorytm Verleta jest stabilny, jeżeli

$$|g_\pm| = \tfrac{1}{2}\left|2+\lambda \pm \sqrt{4\lambda+\lambda^2}\right| < 1\,. \tag{28}$$

Rozważmy trzy przypadki:

1. $-4 \leqslant \lambda \leqslant 0$. Wówczas $4\lambda+\lambda^2 \leqslant 0$.

$$\begin{aligned} g_\pm &= 1+\tfrac{1}{2}\lambda \pm \tfrac{i}{2}\sqrt{\left|4\lambda+\lambda^2\right|}\,, \\ |g_\pm|^2 &= = 1+\lambda+\tfrac{1}{4}\lambda^2+\tfrac{1}{4}\left|4\lambda+\lambda^2\right| \\ &= 1+\lambda+\tfrac{1}{4}\lambda^2-\tfrac{1}{4}(4\lambda+\lambda^2) \equiv 1\,. \end{aligned}$$

   Algorytm jest zatem *neutralnie (marginalnie) stabilny*, co oznacza, że błędy nie są co prawda tłumione, ale też nie narastają i nie prowadzą do rozbieżności.

2. $\lambda > 0$. Wówczas $g_+ > 1$.

3. $\lambda < -4$. Wówczas $g_- < -1$.

Rekapitulując, dla ruchu jednowymiarowego *klasyczny algorytm Verleta jest* marginalnie stabilny *dla* $da/dx|_{t_n} h^2 \in [-4, 0]$ i niestabilny poza tym przedziałem.

Wady algorytmu Verleta

- Dziwne traktowanie prędkości — prędkość w kroku n znana jest dopiero po obliczeniu położenia w kroku $n+1$. W symulacjach znajomość prędkości nie jest potrzebna do obliczania sił, ale jest potrzebna do obliczania energii kinetycznej, temperatury, ciśnienia itp. Prędkość obliczana jest ze wzoru

$$\mathbf{v}_n = \frac{\mathbf{x}_{n+1} - \mathbf{x}_{n-1}}{2h}\,, \tag{29}$$

- We wzorze (22) *różnica* dwu „dużych” wyrażeń rzędu $O(h^0)$ jest dodawana do „małego” wyrażenia rzędu $O(h^2)$, co może prowadzić do znacznej utraty dokładności numerycznej.

Zauważmy, że wyrażenie (29) jest symetryczne w czasie; próba obliczania $\mathbf{v}_n = (\mathbf{x}_n - \mathbf{x}_{n-1})/h$ łamałaby symetrię odwrócenia w czasie, a przecież równania mechaniki (bez sił zależnych od prędkości) są mikroskopowo odwracalne!

Algorytm *leap–frog* („żabiego skoku”)

Spsobem na obejście (niektórych) trudności z algorytmem Verleta jest *algorytm leap–frog* (Hockney 1970), w którym prędkości obliczane są w czasach pośrednich pomiędzy położeniami[§]:

$$\mathbf{v}\left(t_n + \tfrac{1}{2}h\right) = \mathbf{v}\left(t_n - \tfrac{1}{2}h\right) + h\mathbf{a}_n\,, \tag{30a}$$

$$\mathbf{x}_{n+1} = \mathbf{x}(t_n + h) = \mathbf{x}_n + h\mathbf{v}\left(t_n + \tfrac{1}{2}h\right). \tag{30b}$$

Prędkości obliczane są według wzoru

$$\mathbf{v}_n = \tfrac{1}{2}\left[\mathbf{v}\left(t_n - \tfrac{1}{2}h\right) + \mathbf{v}\left(t_n + \tfrac{1}{2}h\right)\right]. \tag{31}$$

[§]Żaby wcale tak nie skaczą.

Zgodność algorytmu leap–frog

Aby sprawdzić czy algorytm leap–frog jest zgodny, należy z (30) wyeliminować prędkości. Po podstawieniu pierwszego z równań (30) do drugiego dostajemy

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathbf{v}\left(t_n - \tfrac{1}{2}h\right) + h^2\mathbf{a}_n\,. \tag{32}$$

Drugie z równań (30) przesunięte w tył w czasie daje

$$h\mathbf{v}\left(t_n - \tfrac{1}{2}h\right) = \mathbf{x}_n - \mathbf{x}_{n-1}\,. \tag{33}$$

Ostatecznie

$$\mathbf{x}_{n+1} = 2\mathbf{x}_n - \mathbf{x}_{n-1} + h^2\mathbf{a}_n\,. \tag{34}$$

Równanie (34) ma taką samą postać co równanie (22), widzimy zatem, iż *algorytm leap–frog jest algebraicznie równoważny algorytmowi Verleta*, a więc jest zgodny z równaniem (1). Nie oznacza to jednak, iż *numeryczne* rezultaty zastosowania obu tych algorytmów są takie same!

Stabilność algorytmu leap–frog

Wyrażenia na propagację błędu w algorytmie leap–frog mają postać

$$\boldsymbol{\eta}_{n+1/2} = \boldsymbol{\eta}_{n-1/2} + h\,\mathbf{J}\boldsymbol{\varepsilon}_n\,, \tag{35a}$$

$$\boldsymbol{\varepsilon}_{n+1} = \boldsymbol{\varepsilon}_n + h\,\boldsymbol{\eta}_{n+1/2}\,, \tag{35b}$$

gdzie $\mathbf{J}_{ij} = \partial \mathbf{a}_i / \partial \mathbf{x}_j \Big|_{t_n}$. Pod względem propagacji błędu *metoda leap–frog zachowuje się jak metoda pół-niejawna!* Ostatecznie macierz wzmocnienia ma postać

$$\mathbf{G} = \begin{bmatrix} \mathbb{I} & h\,\mathbf{J} \\ h\mathbb{I} & \mathbb{I} + h^2\,\mathbf{J} \end{bmatrix}. \tag{36}$$

Dla przypadku jednowymiarowego otrzymujemy stąd

$$g_{\pm} = \frac{2 + \lambda \pm \sqrt{4\lambda + \lambda^2}}{2}\,, \tag{37}$$

gdzie, jak poprzednio, $\lambda = da/dx|_{t_n} h^2$. Jest to identyczne z analogicznym wyrażeniem dla metody Verleta¶, a zatem własności stabilności metody leap-frog i metody Verleta są — co najmniej w przypadku jedowymiarowym — identyczne.

¶Co nie dziwi wobec *algebraicznej* równoważności tych metod.

Cechy algorytmu leap–frog:

- ☺ Różnica dwu dużych wielkości nie jest porównywana z małą.
- ☺ Prędkość w chwili t_n jest znana w chwili t_n, nie dopiero w chwili t_{n+1}.
- ☹ Algorytm wymaga inicjalizacji (metodą RK odpowiedniego rzędu z krokiem połówkowym).
- ☹ Przyspieszenia (siły) nie mogą zależeć od prędkości.
- ☹ Prędkości nadal obsługiwane są dość dziwacznie.

Algorytm *velocity Verlet*

Problemy z dziwacznym tarktowaniem prędkości rozwiązuje kolejna modyfikacja algorytmu Verleta, zwana *velocity Verlet* (Swope et. al. 1982):

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\,\mathbf{v}_n + \tfrac{1}{2}h^2\,\mathbf{a}_n\,, \tag{38a}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + \tfrac{1}{2}h\left(\mathbf{a}_n + \mathbf{a}_{n+1}\right). \tag{38b}$$

Algorytm ten nie nadaje się do sił (przyspieszeń) zależnych od prędkości.

Uogólniony algorytm *velocity Verlet*

Okazuje się, że algorytm *velocity Verlet* można łatwo uogólnić (PFG):

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\,\mathbf{v}_n + \alpha h^2\,\mathbf{a}_n\,, \tag{39a}$$

$$\mathbf{v}_{n+1} = \mathbf{v}_n + h\left(\alpha\,\mathbf{a}_n + (1-\alpha)\mathbf{a}_{n+1}\right), \tag{39b}$$

gdzie $\alpha \in [0, 1]$.

Zgodność algorytmu (39) z równaniem (1) pokażemy eleiminując z (39) prędkości. Odejmując od pierwszego z równań (39) to samo równanie cofnięte o krok w czasie, dostajemy

$$\mathbf{x}_{n+1} = 2\mathbf{x}_n - \mathbf{x}_{n-1} + h\left(\mathbf{v}_n - \mathbf{v}_{n-1}\right) + \alpha h^2\left(\mathbf{a}_n - \mathbf{a}_{n-1}\right). \qquad (40)$$

Z kolei na mocy drugiego z równań (39),

$$\mathbf{v}_n - \mathbf{v}_{n-1} = h\left(\alpha\,\mathbf{a}_{n-1} - (1-\alpha)\mathbf{a}_n\right). \qquad (41)$$

Ostatecznie

$$\begin{aligned} \mathbf{x}_{n+1} &= 2\mathbf{x}_n - \mathbf{x}_{n-1} + h^2\left(\alpha\,\mathbf{a}_n - \alpha\,\mathbf{a}_{n-1} + \alpha\,\mathbf{a}_{n-1} + (1-\alpha)\mathbf{a}_n\right) \\ &= 2\mathbf{x}_n - \mathbf{x}_{n-1} + h^2\mathbf{a}_n\,, \end{aligned} \tag{42}$$

a więc znów dostajemy algebraiczną równoważność (uogólnionego) algorytmu velocity Verlet i klasycznego algorytmu Verleta.

Propagacja błędu dla sił niezależnych od prędkości ma postać

$$\begin{aligned} \varepsilon_{n+1} &= \varepsilon_n + h\boldsymbol{\eta}_n + \alpha h^2\,\mathbf{J}\varepsilon_n\,, && \text{(43a)} \\ \boldsymbol{\eta}_{n+1} &= \boldsymbol{\eta}_n + h\alpha\,\mathbf{J}\varepsilon_n + h(1-\alpha)\widetilde{\mathbf{J}}\varepsilon_{n+1}\,, && \text{(43b)} \end{aligned}$$

$\mathbf{J}$ jest jakobianem w kroku n, $\widetilde{\mathbf{J}}$ jest jakobianem w kroku $n+1$. Jako macierz wzmocnienia dostajemy

$$\mathbf{G} = \begin{bmatrix} \mathbb{I} + \alpha h^2\,\mathbf{J} & h\mathbb{I} \\ \alpha h\,\mathbf{J} + (1-\alpha)h\widetilde{\mathbf{J}}(\mathbb{I} + \alpha h^2\,\mathbf{J}) & \mathbb{I} + (1-\alpha)h^2\,\widetilde{\mathbf{J}} \end{bmatrix}. \qquad (44)$$

W przypadku jedowymiarowym, niezależnie od wartości parametru α, dostajemy takie same współczynniki wzmocnienia, jak w metodach Verleta i leap–frog, a zatem własności stabilności uogólnionej metody velocity Verlet są takie same, jak i innych metod Verleta.

Dla $\alpha = 1$ algorytm (39) można stosować gdy przyspieszenia zależą od prędkości!

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\,\mathbf{v}_n + h^2\,\mathbf{a}_n\,, \tag{45a}$$
$$\mathbf{v}_{n+1} = \mathbf{v}_n + h\,\mathbf{a}_n\,, \tag{45b}$$

co można też zapisać jako

$$\mathbf{v}_{n+1} = \mathbf{v}_n + h\,\mathbf{a}_n\,, \tag{46a}$$
$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\,\mathbf{v}_{n+1}\,. \tag{46b}$$

W tym wypadku macierz wzmocnienia ma postać

$$\mathbf{G} = \begin{bmatrix} \mathbb{I} + h^2\,\mathbf{J}_\mathbf{x} & h\,\mathbb{I} + h^2\,\mathbf{J}_\mathbf{v} \\ h\,\mathbf{J}_\mathbf{x} & \mathbb{I} + h\,\mathbf{J}_\mathbf{v} \end{bmatrix}, \tag{47}$$

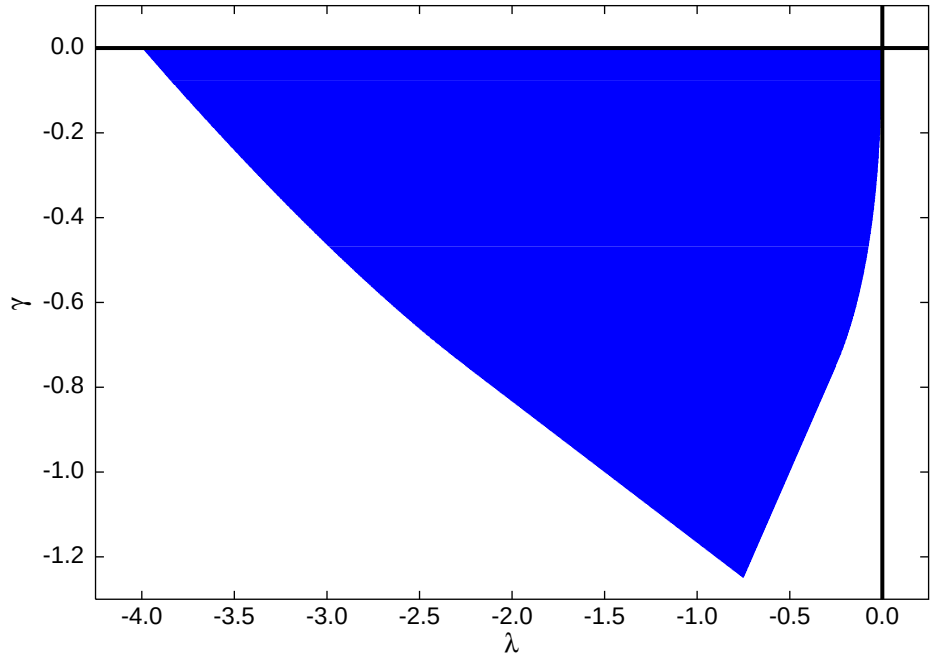
gdzie $\mathbf{J_x}$, $\mathbf{J_v}$ oznaczają, odpowiednio, jakobiany po położeniach i prędkościach.

W przypadku jednowymiarowym dostajemy

$$g_\pm = \tfrac{1}{2}\left(2 + \lambda + \gamma \pm \sqrt{4\lambda + (\lambda+\gamma)^2}\right), \tag{48}$$

gdzie $\lambda = \partial a/\partial x|_{t_n} h^2$, $\gamma = \partial a/\partial v|_{t_n} h$. Dla $\gamma = 0$ (przyspieszenia nie zależą od prędkości) daje to oczywiście to samo, co inne metody Verleta, czyli marginalną stabilność dla $\lambda \in [-4, 0], \gamma = 0$. Dla $\gamma < 0$ i $\lambda < 0$ otrzymujemy ograniczony obszar absolutnej stabilności.

Obszar stabilności dla uogólnionej metody *velocity Verlet* w przypadku jednowymiarowym



Inna modyfikacja algorytmu *velocity Verlet*

Opublikowano kilka modyfikacji algorytmu Verleta przeznaczonych do rozwiązywania równań z przyspieszeniami (siłami) zależnymi od prędkości. Najczęściej używana pochodzi z pracy R. D. Groot, P. B. Warren, *Dissipative particle dynamics: Bridging the gap between atomistic and mesoscopic simulations*, J. Chem. Phys. **107**, 4423 (1997). Jest to algorytm typu *predyktor-korektor*:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\mathbf{v}_n + \tfrac{1}{2}h^2\mathbf{a}_n\,, \tag{49a}$$

predyktor:

$$\widetilde{\mathbf{v}} = \mathbf{v}_n + \beta h\mathbf{a}_n\,, \tag{49b}$$

$$\widetilde{\mathbf{a}} = \mathbf{a}\left(\mathbf{x}_{n+1}, \widetilde{\mathbf{v}}\right), \tag{49c}$$

korektor:

$$\mathbf{v}_{n+1} = \mathbf{v}_n + \tfrac{1}{2}h\left(\mathbf{a}_n + \widetilde{\mathbf{a}}\right), \tag{49d}$$

gdzie $\beta \in [0,1]$. W przeciwieństwie do algorytmu (45), algorytm (49) wymaga dwu obliczeń przyspieszenia na krok czasowy.

Algorytmy Verleta są symplektyczne

Twierdzenie: Jeżeli hamiltonian jest separowalny, algorytm Verleta i inne algorytmy równoważne mu algebraicznie (*leap-frog*, *velocity Verlet*) są symplektyczne.