

Zápisky k předmětu Využití počítačů ve fyzice

Pavel Stránský

24. března 2020

1 Obyčejné diferenciální rovnice

Každou obyčejnou diferenciální rovnici n -tého řádu lineární v nejvyšší derivaci lze převést na soustavu n obyčejných diferenciálních rovnic prvního řádu ve tvaru

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, t), \quad (1)$$

kde $\mathbf{x} = \mathbf{x}(t)$ je vektor hledaných funkcí.

Příklad 1.1: *Pohybovou rovnici*

$$Ma = F(y), \quad (2)$$

kde M je hmotnost pohybujícího se tělesa, $y = y(t)$ jeho poloha a $a = a(t) = d^2y/dt^2$ převedeme na dvě diferenciální rovnice prvního řádu triviálním zavedením rychlosti $v = v(t) = dy/dt$:

$$\frac{d}{dt} \begin{pmatrix} y \\ v \end{pmatrix} = \begin{pmatrix} v \\ \frac{1}{M}F(y) \end{pmatrix}, \quad (3)$$

tj. vektor funkce pravých stran podle rovnice (1) je

$$\mathbf{f}(\mathbf{x}, t) = \begin{pmatrix} v \\ \frac{1}{M}F(y) \end{pmatrix} \quad (4)$$

kde $\mathbf{x} \equiv (y, v)$.

Příklad 1.2: *Pohybová rovnice pro harmonický oscilátor (matematické kyvadlo s malou výchylkou) při volbě jednotek $M = \Omega = 1$, kde M je hmotnost kmitající částice a Ω její rychlost, zní*

$$a = \frac{d^2y}{dt^2} = -y \quad \Longleftrightarrow \quad \frac{d}{dt} \begin{pmatrix} y \\ v \end{pmatrix} = \begin{pmatrix} v \\ -y \end{pmatrix} \quad (5)$$

Úkol 1.1: *Převed'te na soustavu obyčejných diferenciálních rovnic prvního řádu rovnici třetího řádu pro Hiemenzův tok*

$$y''' + yy'' - y'^2 + 1 = 0. \quad (6)$$

Řešení 1.1: *Hledaná soustava diferenciálních rovnic je*

$$\frac{d}{dt} \begin{pmatrix} y \\ v \\ a \end{pmatrix} = \begin{pmatrix} v \\ a \\ -ya + v^2 - 1 \end{pmatrix}. \quad (7)$$

1.1 Diferenciální rovnice prvního řádu

Drtivá většina knihoven a algoritmů pro integraci diferenciálních rovnic počítá s rovnicemi ve tvaru (1). Zde se omezíme na jednu rovnici

$$\frac{dy}{dt} = f(y, t), \quad (8)$$

přičemž rozšíření na soustavu je triviální: místo skalárů y a f vezmeme vektory.

Řešení diferenciální rovnice spočívá v nahrazení infinitezimálních přírůstků přírůstky konečnými:

$$\frac{\Delta y}{\Delta t} = \phi(y, t) \quad (9)$$

kde ϕ je funkce, která udává směr, podél kterého se při numerickém řešení vydáme. Volba této funkce je klíčová a záleží na ní, jak přesné řešení dostaneme a jak rychle ho dostaneme.

1.1.1 Pár důležitých pojmů

- **Jednokrokové algoritmy:** Algoritmy, které výpočtu následujícího kroku hodnoty funkce y_{i+1} vyžadují znalost pouze aktuálního kroku y_i . Rozepsáním (9) dostaneme

$$y_{i+1} = y_i + \underbrace{\phi(y_i, t)}_{\phi_i} \Delta t, \quad (10)$$

přičemž počáteční hodnota y_0 je dána počáteční podmínkou. My se omezíme pouze na tyto algoritmy.

- **Lokální diskretizační chyba:**

$$\mathcal{L} = y(t + \Delta t) - y(t) - \phi(y(t), t)\Delta t, \quad (11)$$

kde $y(t)$ udává přesné řešení v čase t .

- **Akumulovaná diskretizační chyba:**

$$\epsilon_i = y_i - y(t_i) \quad (12)$$

- **Řád metody:** Metoda je p -tého řádu, pokud

$$L(\Delta t) = \mathcal{O}(\Delta t^{p+1}) \quad (13)$$

- **Symplektické algoritmy:** Speciální algoritmy navržené pro řešení pohybových diferenciálních rovnic. Od běžných algoritmů je odlišuje to, že zachovávají objem fázového prostoru, a tedy i energii (zatímco u obecných algoritmů energie s integračním časem roste). V praxi se ze symplektických algoritmů používá pouze Verletův algoritmus 1.3.
- **Kontrola chyby řešení:** Chybu numerického řešení diferenciální rovnice lze zmenšit 1) menším krokem, 2) lepší metodou (metodou vyššího řádu). Menší krok však znamená vyšší výpočetní čas. Sofistikované metody proto průběžně mění velikost kroku: když se funkce mění pomalu, krok prodlouží, když se mění rychle, krok zkrátí (tzv. **metody s adaptivním krokem**). Tím se docílí vysoké přesnosti při co nejmenším výpočetním čase.

1.1.2 Eulerova metoda 1. řádu

$$\phi_i = f(y_i, t_i), \quad (14)$$

tj. krok do y_{i+1} děláme vždy ve směru tečny v bodě y_i .

- Nejjednodušší metoda integrace diferenciálních rovnic.
- Chyba je obrovská, k dosažení přesných hodnot je potřeba velmi malého kroku, což znamená dlouhý výpočetní čas.

1.1.3 Eulerova metoda 2. řádu

$$\begin{aligned} k_1 &= f(y_i, t_i) \\ k_2 &= f(y_i + k_1 \Delta t, t + \Delta t) \\ \phi_i &= \frac{1}{2} (k_1 + k_2), \end{aligned} \quad (15)$$

tj. uděláme jednoduchý Eulerův krok ve směru k_1 , spočítáme derivaci k_2 po tomto kroku a vyrazíme z bodu y_i ve směru, který je průměrem obou směrů (doporučuji si nakreslit obrázek).

Ekvivalentní je udělat „Eulerův půlkrok“ a vyrazit z bodu y_i ve směru derivace spočtené po tomto půlkroku:

$$\begin{aligned} k'_1 &= f(y_i, t_i) \\ k'_2 &= f\left(y_i + k'_1 \frac{\Delta t}{2}, t + \frac{\Delta t}{2}\right) \\ \phi_i &= k'_2 \end{aligned} \quad (16)$$

1.2 Runge-Kuttova metoda 4. řádu

$$\begin{aligned} k_1 &= f(y_i, t_i) \\ k_2 &= f\left(y_i + k_1 \frac{\Delta t}{2}, t + \frac{\Delta t}{2}\right) \\ k_3 &= f\left(y_i + k_2 \frac{\Delta t}{2}, t + \frac{\Delta t}{2}\right) \\ k_4 &= f(y_i + k_3 \Delta t, t + \Delta t) \\ \phi_i &= \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4) \end{aligned} \quad (17)$$

- Jedna z nejčastěji používaných metod.
- Vysoká rychlost a přesnost při relativní jednoduchosti.
- Existují i Runge-Kuttovy metody vyššího řádu p , avšak vyžadují výpočet více než p dílčích derivací k_j . Obecně platí, že metoda řádu $p \leq 4$ vyžaduje p derivací, metoda řádu $5 \leq p \leq 7$ vyžaduje $p + 1$ derivací a metoda řádu $p = 8, 9$ vyžaduje $p + 2$ derivací.

1.3 Verletova metoda

Pro rovnici 2. řádu ve tvaru (pohybovou rovnici)

$$M \frac{d^2 y}{dt^2} = F(y), \quad (18)$$

kde M je hmotnost pohybující se částice a F síla, která na ni působí. Algoritmus je

$$\begin{aligned} y_{i+1} &= y_i + v_i \Delta t + \frac{1}{2} a_i \Delta t^2, \\ v_{i+1} &= v_i + \frac{1}{2} (a_{i+1} + a_i) \Delta t, \end{aligned} \quad (19)$$

kde $a_i \equiv F(y_i)/M$.

- Symplektický algoritmus, tj. algoritmus zachovávající energii (pokud systém popsaný rovnicí (18) energii zachovává).
- Užívá se nejčastěji v molekulární dynamice k simulaci pohybu velkého množství vzájemně interagujících částic.
- Řád této metody je $p = 2$. Symplektické algoritmy s vyšším řádem existují, avšak v praxi se nepoužívají.

Úkol 1.2: *Naprogramujte Eulerovu metodu 1. a 2. řádu¹, Runge-Kuttovu metodu a Verletovu metodu a vyřešte diferenciální rovnici harmonického oscilátoru*

$$\frac{d^2 y}{dt^2} = -y \quad (20)$$

s počátečními podmínkami $y_0 = 0$, $y'_0 \equiv v_0 = 1$ (analytickým řešením je funkce $\sin t$). Časový krok ponechte jako volný parametr. Nakreslete grafy řešení $y(t)$ a grafy energie $E(t)$ pro rozdílné hodnoty integračních kroků, například $\Delta t = 0.01$ a $\Delta t = 0.1$ pro čas $t \in \langle 0; 30 \rangle$. Energie harmonického oscilátoru je dána vzorcem

$$E = \frac{1}{2} (y^2 + v^2). \quad (21)$$

Přesvědčte se, že jediná Verletova metoda skutečně zachovává energii. Pro ostatní metody energie roste.

Řešení 1.2: *Jedno možné řešení je rozděleno do dvou souborů `ODE.py` a `Oscillator.py`, které můžete stáhnout z GitHubu <https://www.github.com/PavelStransky/PCInPhysics> (o GitHubu bude pojednáno v sekci ??).*

- **ODE.py:** Modul napsaný dostatečně obecně, aby ho bylo možné použít na řešení libovolných soustav diferenciálních rovnic.
 - **StepEuler1:** Integrační krok Eulerovy metody 1. řádu.
 - **StepEuler2:** Integrační krok Eulerovy metody 2. řádu.
 - **StepVerlet:** Integrační krok Verletovy metody.
 - **StepRungeKutta:** Integrační krok Runge-Kuttovy metody 4. řádu.
 - **ODESolution:** Integruje diferenciální rovnici danou pravými stranami prvního parametru **Derivatives** pomocí kroku (metody) dané parametrem **Step** a délkou **dt**, přičemž vrátí pole hodnot řešení soustavy rovnic v jednotlivých časech, pole časů a jméno integračního kroku (pro snazší pozdější porovnání různých metod).

¹Tyto metody jsme již naprogramovali na cvičení minulý týden.

- **ScipyODESolution**: Integruje diferenciální rovnici pomocí funkce `odeint` z knihovny `scipy.integrate`. Pozor, parametr `dt` zde neznamena integrační krok, nýbrž časový krok výsledného pole. Funkce `odeint` používá sofistikovaný řešitel diferenciálních rovnic s proměnným krokem. Pro více detailů můžete mrknout na dokumentaci k této funkci na <https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html>.
- **ShowGraphSolutions**: Vykreslí graf řešení diferenciální rovnice (jako první parametr `odeSolutions` lze zadat seznam více řešení různými metodami či s různým krokem). Parametr `ExactFunction` je odkaz na přesné řešení dané diferenciální rovnice. Je-li specifikován, vykreslí se do grafu dva panely: jeden s hodnotami numerického řešení, druhý s rozdílem řešení numerického a přesného.

Funkce `StepEuler1`, `StepEuler2` a `StepRungeKutta` fungují pro obecnou soustavu n obyčejných diferenciálních rovnic prvního řádu. Funkce `StepVerlet` funguje jen pro pohybovou rovnici, tj. pro jednu diferenciální rovnici původně druhého řádu.

- **Oscillator.py**: Soubor, který využívá obecných funkcí z modulu `ODE.py` pro integraci harmonického oscilátoru různými metodami.
 - **Energy**: Pro zadanou polohu a rychlost vrátí energii harmonického oscilátoru.
 - **Derivatives**: Pravá strana soustavy diferenciálních rovnic harmonického oscilátoru.
 - **CompareMethods**: Vyřeší diferenciální rovnici harmonického oscilátoru různými metodami a řešení nakreslí do jednoho grafu. Následně vykreslí grafy $E(t)$. Harmonický oscilátor je konzervativní systém (zachovává energii), rostoucí energie je způsobena nepřesností integrační metody.

Príslušné grafy jsou zobrazeny v obrázku 1.2.

Úkol 1.3: Rozšířte kód tak, aby počítal průměrnou kumulovanou chybu

$$\mathcal{E} = \sqrt{\frac{1}{n} \sum_{i=0}^{n-1} (y_i - \sin t_i)^2} \quad (22)$$

a nakreslete závislost $E(\Delta t)$ pro $\Delta t \in \langle 0.002; 0.1 \rangle$ a pro různé metody. Jelikož očekáváme mocninovou závislost dle (13), kde exponent je tím větší, čím větší je řád metody, je výhodné graf $E(\Delta t)$ kreslit v log-log měřítku. V Pythonu použijete místo `plot(...)` funkci `loglog(...)` z knihovny `matplotlib.pyplot`.

Řešení 1.3: Průměrnou kumulovanou chybu počítá funkce `CumulativeError` z modulu `ODE.py`. Srovnání řešení diferenciální rovnice harmonického oscilátoru různými integračními metodami zakresluje do grafu funkce `ShowGraphCumulativeErrors` ze souboru `Oscillator.py`. Výsledný graf je znázorněn na obrázku 1.3. Vypočítanými křivkami $\ln \mathcal{E}(\ln \Delta t)$ je proložena přímka, jejíž sklon α udává exponent mocninného zákona

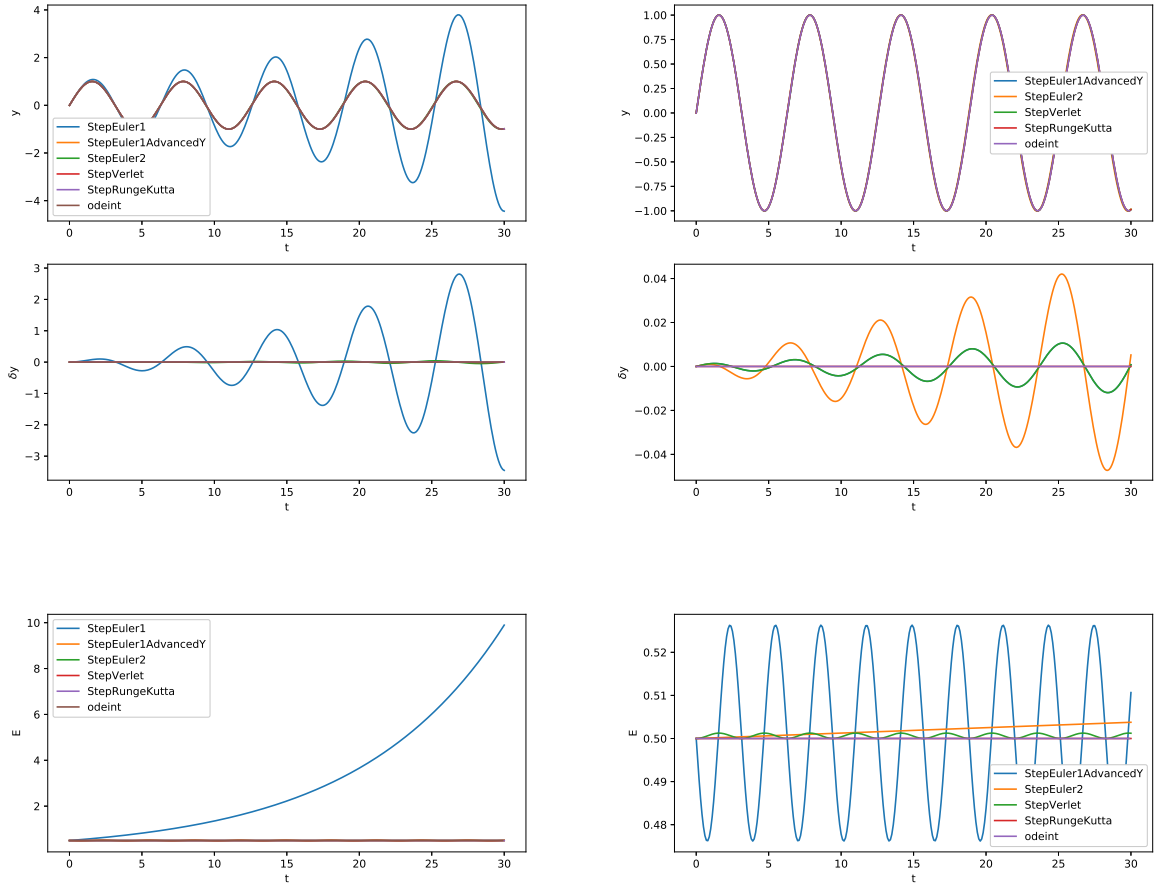
$$\mathcal{E} \propto (\Delta t)^\alpha \quad (23)$$

(k proložení přímky je využita funkce pro lineární regresi `linregress` z knihovny `scipy.stats`). Tento exponent výborně odpovídá řádu metody p dle rovnice (13).

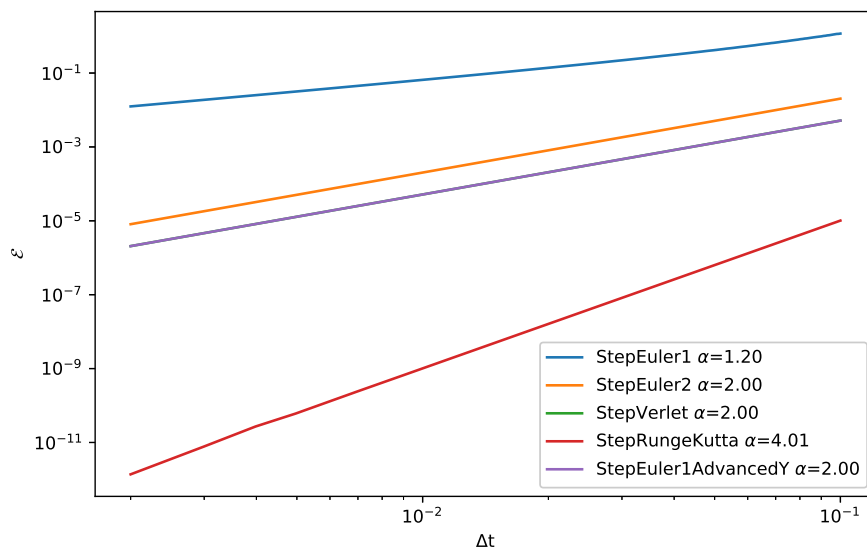
Úkol 1.4: Eulerovu metodu 1. řádu lze pro harmonický oscilátor vylepšit následující záměnou:

$$\begin{aligned} y_{i+1} &= y_i + v_i \Delta t \\ v_{i+1} &= v_i - y_i \Delta t \end{aligned} \quad \longrightarrow \quad \begin{aligned} y_{i+1} &= y_i + v_i \Delta t \\ v_{i+1} &= v_i - y_{i+1} \Delta t \end{aligned} \quad (24)$$

(vypočítáme y_{i+1} a tuto hodnotu použijeme namísto hodnoty y_i pro výpočet rychlosti v_{i+1}). Naprogramujte tuto metodu a ukažte, že pro harmonický oscilátor se jedná o metodu 2. řádu. Využijte srovnání v grafu z předchozí úlohy.



Obrázek 1: Integrace diferenciální rovnice harmonického oscilátoru (20) různými metodami. Časový krok je $\Delta t = 0.1$. *Levý sloupec:* všechny metody. *Pravý sloupec:* bez Eulerovy metody 1. řádu. 1. řádek: hodnoty $y(t)$. 2. řádek: rozdíly $\delta y(t) = y(t) - \sin t$. Pro Eulerovu metodu 1. řádu je divergence numerického od analytického řešení očividně exponenciální v čase. 3. řádek: energie (21). Pro Eulerovy metody energie roste. Energie roste i pro Runge-Kuttovu metodu a pro integraci pomocí funkce `odeint`, avšak růst je řádově menší, tudíž není na grafech při daném měřítku svislé osy vidět. Naopak pro Verletův algoritmus a pro „předbíhající“ Eulerovu metodu energie osciluje okolo počáteční energie $E = \frac{1}{2}$.



Obrázek 2: Závislost průměrné kumulované chyby (22) na délce kroku Δt vypočítaná a vykreslená pomocí funkce `ShowGraphCumulativeErrors` pro harmonický oscilátor (soubor `Oscillator.py`). Křivka pro Verletovu metodu je „schovaná“ za křivkou pro předbíhající Eulerovu metodu.

Řešení 1.4: Tato metoda je naimplementována v modulu `ODE.py` funkcemi `StepEuler1AdvancedY` a `StepEuler1AdvancedV`. Ze srovnání s ostatními metodami zobrazené v obrázcích 1.2 a 1.3 vyplývá, že tato metoda je

- symplektická (energie sice osciluje a osciluje s větší amplitudou než pro Verletovu metodu, ale pořád osciluje okolo počáteční hodnoty),
- 2. řádu.

V obrázcích jsou výsledky pouze pro metodu předbíhající v souřadnici. Díky symetrii jsou výsledky pro metodu předbíhající v rychlosti identické.

Úkol 1.5: Využijte hotové kódy a pohrajte si s řešením rovnice pro klesající exponenciálu

$$\frac{d^2 y}{dt^2} = y \quad (25)$$

s počátečními podmínkami $y_0 = 1$, $y'_0 = -1$. Přesvědčte se, že Verletova metoda a vylepšená Eulerova metoda z posledního bodu jsou nestabilní — pro tuto rovnici v relativně krátkém čase začnou řešení exponenciálně divergovat.

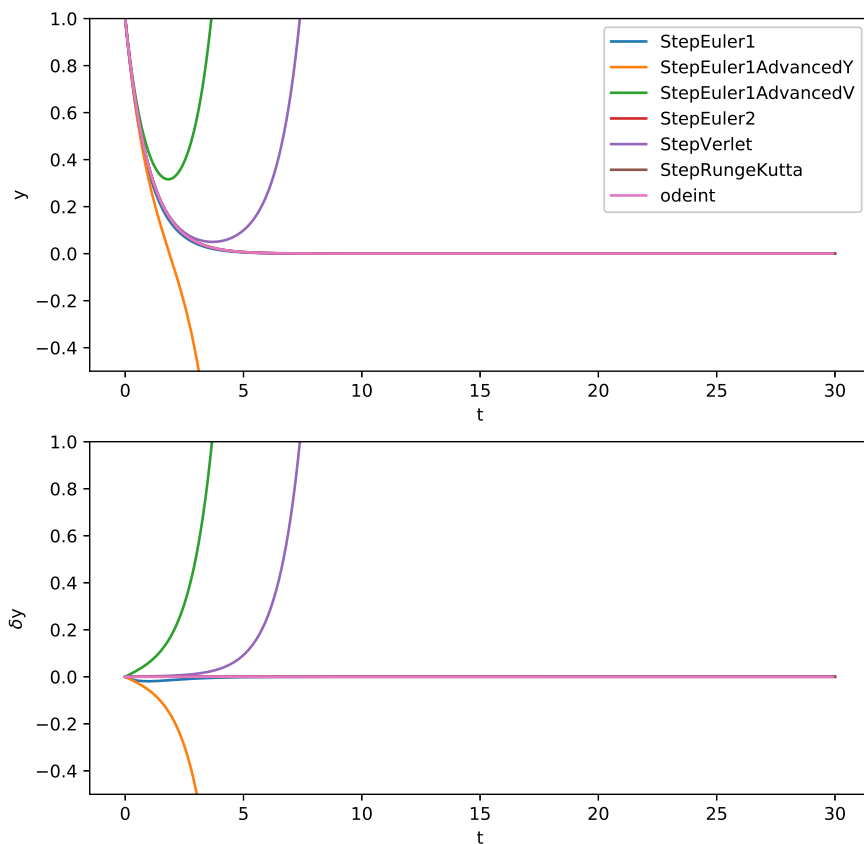
Řešení 1.5: Řešení analogické k příkladu harmonického oscilátoru je v souboru `Exp.py`. Tento systém není konzervativní — nelze nadefinovat zachovávající se veličinu, která by měla význam energie.

Z obrázku 1.5 je vidět, že symplektické algoritmy jsou zde nestabilní, a tudíž nejsou na tento typ úlohy vhodné, což je pochopitelné, protože symplektické algoritmy jsou navrženy pouze pro energii zachovávající systémy. Obecné řešení rovnice (25) má tvar

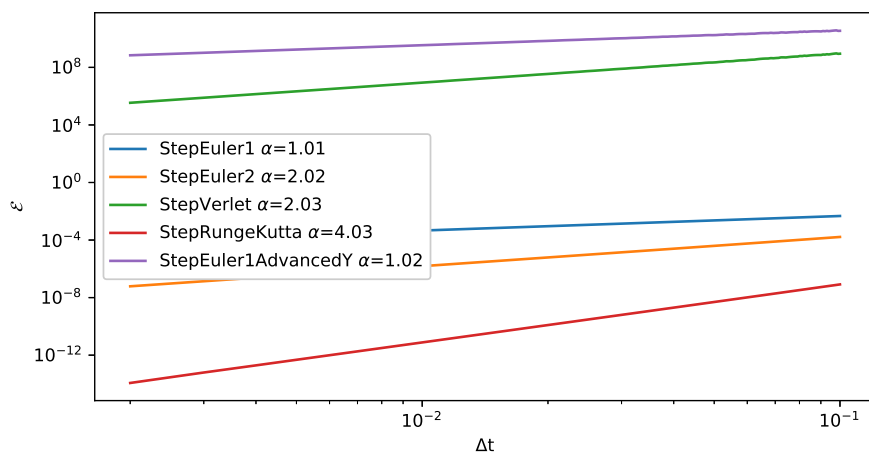
$$y(t) = A e^t + B e^{-t}, \quad (26)$$

přičemž my speciálními počátečními podmínkami vybíráme pouze exponenciálně klesající řešení. Symplektické algoritmy v určité chvíli „překmitnou“ na exponenciálně rostoucí řešení a začnou divergovat.

Průměrná kumulovaná chyba je znázorněna na obrázku 1.5.



Obrázek 3: Totéž jako v obrázku 1.2, avšak pro exponenciálně klesající systém daný rovnicí (25). Symplektické algoritmy jsou nestabilní.



Obrázek 4: Totéž jako v obrázku 1.3, avšak pro exponenciálně klesající systém daný rovnicí (25). Pro symplektické algoritmy uvedený graf nedává příliš smysl, jelikož chyba je o řády vyšší než hledané řešení. Přesto stojí za povšimnutí, že předbíhající Eulerův algoritmus, který výborně funguje pro harmonický oscilátor a je pro něj metodou 2. řádu, se zde chová jako metoda 1. řádu.

1.4 Shrnutí

- Řešitelé obyčejných diferenciálních rovnic převážně pracují se soustavami diferenciálních rovnic prvního řádu. Na tento tvar není obtížné diferenciální rovnici vyššího řádu převést.
- Nejčastěji se používají jednokrokové metody, jejichž hlavní výhoda je v možnosti jednoduše měnit délku kroku.
- Přesnost řešení závisí na řádu metody p a na délce integračního kroku Δt . Čím je řád metody vyšší, tím rychleji klesá chyba se zmenšujícím se krokem. V praxi, pokud nechcete svěřit svůj problém černé skřínce ve formě nějaké hotové knihovny, se velmi často používá Runge-Kuttova metoda 4. řádu, která je jednoduchá na implementaci, je stabilní a rychlá.
- Symplektické metody, z nichž nejběžnější je Verletova metoda, jsou výhodné k modelování fyzikálních systémů zachovávajících energii. Pro nekonzervativní systémy nejsou vhodné.
- V Pythonu se procvičilo:
 - předávání odkazů na funkce v argumentu,
 - vracení více hodnot z funkce a jejich následné zpracování,
 - vykreslování grafů a práce s více panely pomocí funkce `subplot`; argument této funkce přijímá trojciferné číslo, ve kterém první cifra udává počet panelů v řádcích, druhá cifra počet panelů ve sloupcích a třetí cifra pořadové číslo vykreslovaného panelu,
 - funkce `lineregression` z knihovny `scipy.stats` pro výpočet lineární regrese.

A nyní již umíte vypočítat a nakreslit průběh funkce sinus :-)