Zápisky k předmětu Využití počítačů ve fyzice

Pavel Stránský

14. dubna 2020

Obsah

1 Obyčejné diferenciální rovnice **1**
1.1 Diferenciální rovnice prvního řádu . . . 2
1.1.1 Pár důležitých pojmů . . . 2
1.1.2 Eulerova metoda 1. řádu . . . 3
1.1.3 Eulerova metoda 2. řádu . . . 3
1.2 Runge-Kuttova metoda 4. řádu . . . 4
1.3 Verletova metoda . . . 4
1.4 Shrnutí . . . 9

2 Git (pokračování) **10**
2.1 Vzdálené repozitáře . . . 10
2.2 .gitignore . . . 11

3 Náhodná procházka **12**
3.1 Pseudonáhodná čísla . . . 12

4 Hledání minima funkce **15**
4.1 Metropolisův algoritmus . . . 18
4.2 Minimalizace pomocí knihovny SciPy . . . 20
4.3 Shrnutí . . . 20

5 LaTeX **20**
5.1 Formát TEXovského souboru . . . 21
5.2 Struktura TEXovského souboru . . . 22
5.2.1 Preambule . . . 22
5.2.2 Tělo dokumentu . . . 22
5.3 Další návody a odkazy . . . 23

6 Histogram **23**
6.1 Základní definice a tvrzení z teorie pravděpodobnosti . . . 23
6.2 Příklady náhodných veličin . . . 25
6.3 Výběr z neznámého rozdělení . . . 26

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

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = \boldsymbol{f}(\boldsymbol{x}, t), \tag{1}$$

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

Příklad 1.1: *Pohybovou rovnici*

$$Ma = F(y), \tag{2}$$

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

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} y \\ v \end{pmatrix} = \begin{pmatrix} v \\ \frac{1}{M}F(y) \end{pmatrix}, \tag{3}$$

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

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

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

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*

$$a = \frac{\mathrm{d}^2y}{\mathrm{d}t^2} = -y \qquad \Longleftrightarrow \qquad \frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} y \\ v \end{pmatrix} = \begin{pmatrix} v \\ t \end{pmatrix} \tag{5}$$

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

$$y''' + yy'' - y'^2 + 1 = 0. \tag{6}$$

Řešení 1.1: *Hledaná soustava diferenciálních rovnic je*

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} y \\ v \\ a \end{pmatrix} = \begin{pmatrix} v \\ a \\ -ya + v^2 - 1 \end{pmatrix}. \tag{7}$$

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

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(y, t), \tag{8}$$

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

Řešení diferenciální rovnice spočívá v nahrazení infinitezimálních přírůstků přírůstky konečnými:

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

kde ϕ je funkce, která udává směr, podél kterého se při numerickém řešení vydáme. Volbá 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

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

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

- **Lokální diskretizační chyba:**
$$\mathcal{L} = y(t + \Delta t) - y(t) - \phi(y(t), t)\Delta t, \tag{11}$$
kde $y(t)$ udává přesné řešení v čase t.
- **Akumulovaná diskretizační chyba:**
$$\epsilon_i = y_i - y(t_i) \tag{12}$$
- **Řád metody:** Metoda je p-tého řádu, pokud
$$L(\Delta t) = \mathcal{O}(\Delta t^{p+1}). \tag{13}$$
- **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 se energie s integračním časem mění a většinou 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

$$\phi_i = f(y_i, t_i), \tag{14}$$

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

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

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:

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

1.2 Runge-Kuttova metoda 4. řádu

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

- 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 $p = 8, 9$ vyřaduje $p + 2$ derivací.

1.3 Verletova metoda

Pro rovnici 2. řádu ve tvaru (pohybovou rovnici)

$$
M\frac{\mathrm{d}^2 y}{\mathrm{d}t^2} = F(y), \tag{18}
$$

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

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

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. řádu*[1]*, Runge-Kuttovu metodu a Verletovu metodu a vyřešte diferenciální rovnici harmonického oscilátoru*

$$
\frac{\mathrm{d}^2 y}{\mathrm{d}t^2} = -y \tag{20}
$$

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

[1] Tyto metody jsme již naprogramovali na cvičení minulý týden.

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

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

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 2.1).*

- `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).*
  - `ScipyODESolution`*: Integruje diferenciální rovnici pomocí funkce* `odeint` *z knihovny* `scipy.integrate`*. Pozor, parametr* `dt` *zde neznamená integrační krok, nýbrž časový krok výsledného pole. Funkce* `odeint` *používá sofistikovaný řešitel diferenciálních rovnic s proměnným krokem. Pro podrobnosti můžete mrknout na dokumentaci k této funkci.*
  - `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.*

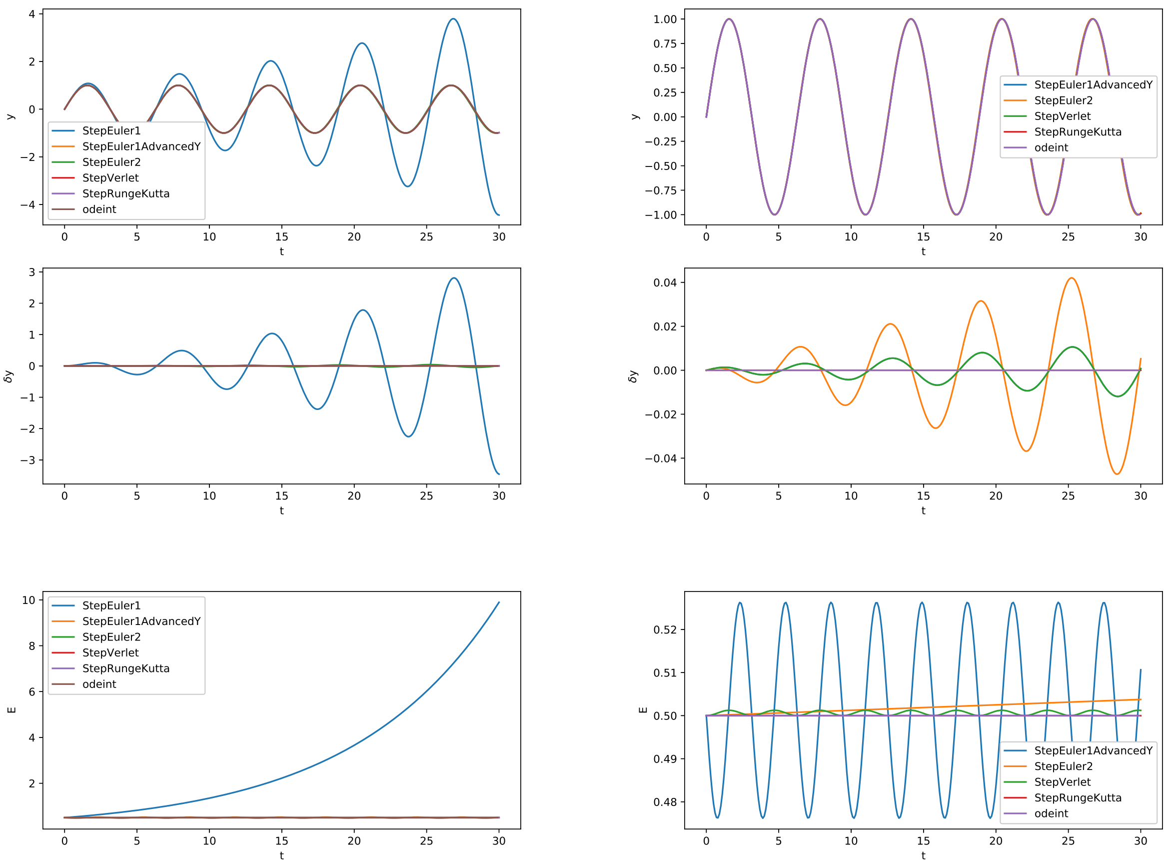
  Funkce `StepEuler1`*,* `StepEuler2` *a* `StepRungeKutta` *fungují pro obecnou soustavu n obyčejných diferenciálních rovnic prvního řádu. Funkce* `StepVerlet` *funguje jen pro pohybovou rovnici, tj. pro jednu diferencální rovnici původně druhého řádu přepsanou na dvě diferenciální rovnice prvního řádu.*

- `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.*

Příslušné grafy jsou zobrazeny v obrázku 1.

Úkol 1.3: *Rozšiřte kód tak, aby počítal průměrnou kumulovanou chybu*

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



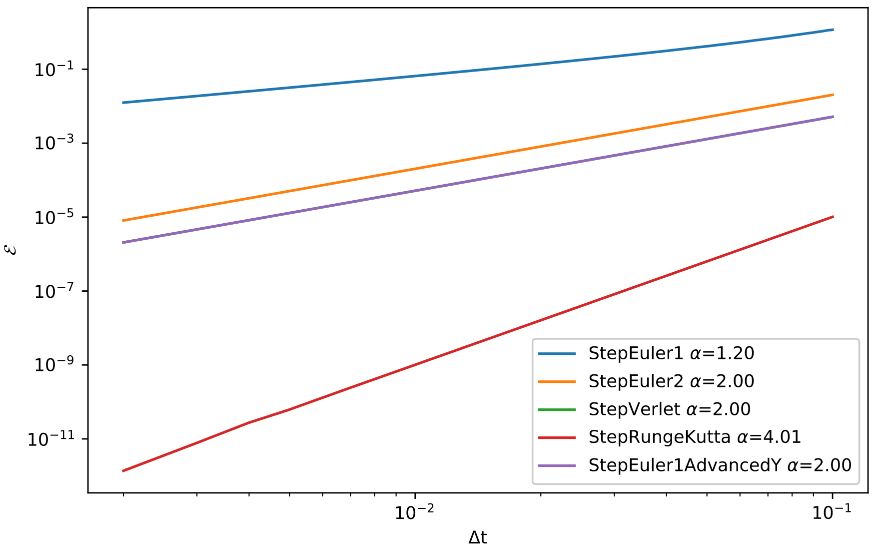
Obrázek 1: Integrace diferenciální rovnice harmonického oscilátoru (20) různými metodami. Časový krok je $\Delta t = 0.1$. *Levý sloupec:* všechny metody. *Pravý sloupec:* bez Eulerovy metody 1. řádu. *1. řádek:* hodnoty $y(t)$. *2. řádek:* rozdíly $\delta y(t) = y(t) - \sin t$. Pro Eulerovu metodu 1.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

a nakreslete závislost $E(\Delta t)$ pro $\Delta t \in \langle 0.002; 0.1\rangle$ a pro různé metody. Jelikož očekáváme mocninnou 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 2. Vypočítanými křivkami $\ln \mathcal{E}(\ln \Delta t)$ je proložena přímka, jejíž sklon α udává exponent moncinného zákona*

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

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



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.

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

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

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

Řešení 1.4: *Tato metoda je naimplementována v modulu* `ODE.py` *funkcemi* `StepEuler1AdvancedY` *a* `StepEuler1AdvancedV`. *Ze srovnání s ostatními metodami zobrazené v obrázcích 1 a 2 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.*

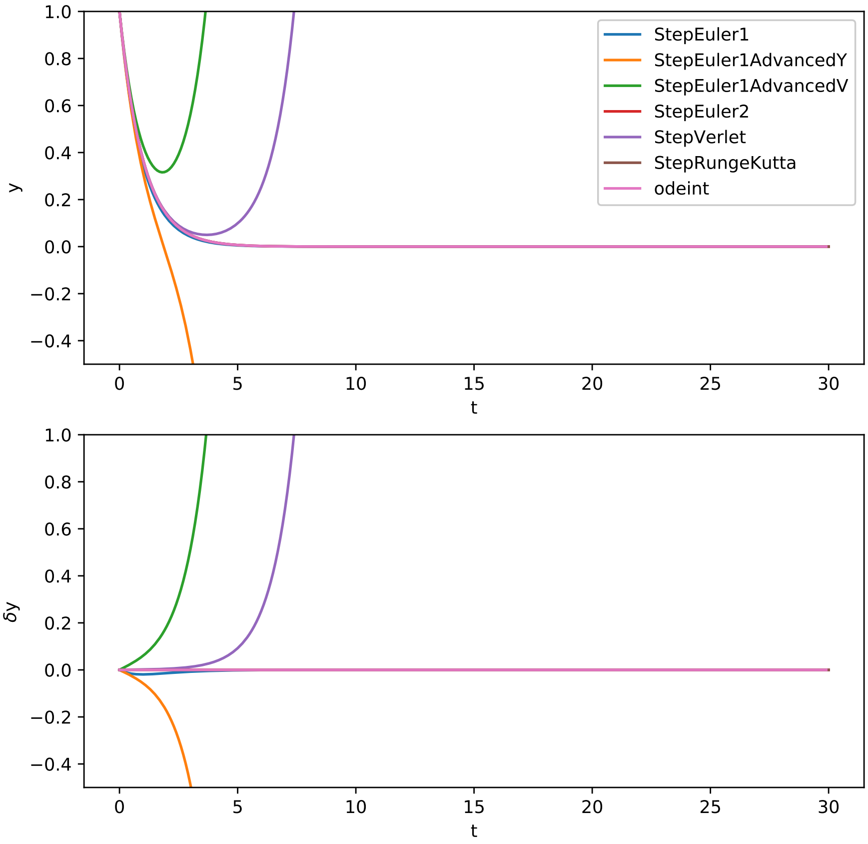
V obrázcích jsou výsledky pouze pro metodu předbíhající v souřadnici. Díky symetrii jsou výsledky pro metodu předbíhající v rychlosti identické.

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

$$\frac{\mathrm{d}^2 y}{\mathrm{d}t^2} = y \tag{25}$$

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.*

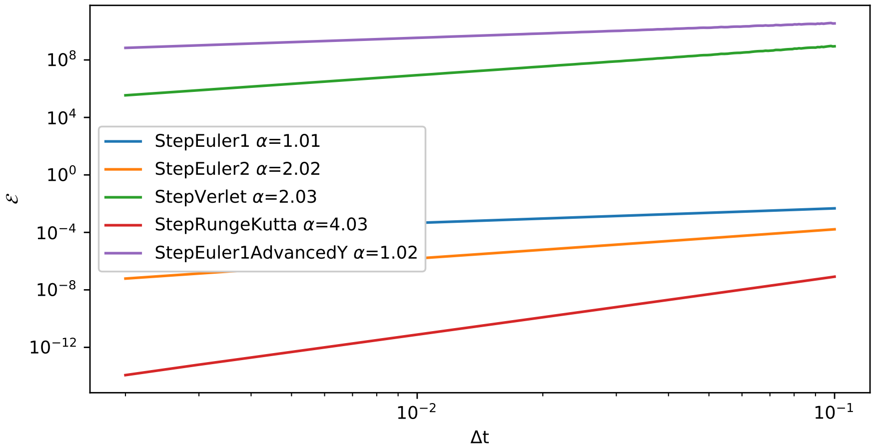


Obrázek 3: Totéž jako v obrázku 1, avšak pro exponenciálně klesající systém daný rovnicí (25). Symplektické algoritmy jsou nestabilní.

Z obrázku 3 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

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

přičemž my speciálními počátečními podmínkami vybíráme pouze exponenciálně klesající řešení. Sym-



Obrázek 4: Totéž jako v obrázku 2, 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.

plektické algoritmy v určitou chvíli „překmitnou“ na exponenciálně rostoucí řešení a začnou divergovat.

Průměrná kumulovaná chyba je znázorněna na obrázku 4.

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 zmenšujícícm se krokem. V praxi, pokud nechcete svěřit svůj probém černé skříňce 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 :-)

2 Git (pokračování)

2.1 Vzdálené repozitáře

Verzovací systém GIT umí kromě sledování a uchovávání historie změn souborů vašeho projektu i koordinovat změny, které provádí na projektu více řešitelů (pracovní tým). K tomu slouží vzdálené repozitáře. Mezi nejrozšířenější systémy patří

- GitHub: Na něm si každý může zdarma zřídit vlastní repozitáře a používat je například k synchronizaci svého projektu mezi různými počítači, pro sdílení vlastních výtvorů s komunitou nebo právě pro spolupráci ve vlastním týmu.
- GitLab: Open source řešení, základní verze rovněž zdarma, k pokročilé verzi má MFF UK licenci.

Oba tyto systémy disponují webovým rozhraním, ze kterého lze projekty jednoduše spravovat, a rovněž desktopovými aplikacemi, které umožňují pracovat s lokálními repozitáři pomocí jednoduchého rozhraní. Pro GitHub existuje například GitHub Desktop, ale i spousta dalších. Projekty mohou být soukromé (přístup k nim máte pouze vy či ti, kterým pošlete pozvánku) nebo veřejné (přístup má kdokoliv).

Tyto zápisky a vzorové ukázky kódů jsou veřejně na GitHubu a můžete k nim dostat na adrese `https://github.com/PavelStransky/PCInPhysics`. K dispozici je samozřejmě celý repozitář s historií se všemi commity a se všemi vývojovými větvemi. Repozitář můžete stáhnout buď z uvedené webové stránky (zelené tlačítko `Clone or download`), nebo pomocí programu git.

- `git clone https://github.com/PavelStransky/PCInPhysics`

  Vytvoří adresář `PCInPhysics` a do něj stáhne celý repozitář. V případě těchto zápisků se jedná o tento soubor v LaTeXu, výsledné PDF, EPS verze všech obrázků a všechny zdrojové kódy.

- `git remote -v`

  V adresáři s lokálním repozitářem ukáže, na jaký vzdálený repozitář je navázaný. Stejně jako základní větev se standardně jmenuje `master`, vzdálený repozitář se standardně jmenuje `origin`. Vy můžete mít na jeden projekt navázáno více vzdálených repozitářů, každý pak samozřejmě musíte pojmenovat jinak.

- `git remote add origin https://github.com/Uzivatel/VzdalenyRepozitar`

  Přidá do vašeho projektu odkaz na vzdálený repozitář.

- `git remote show origin`

  Zobrazí informace o vzdáleném repozitáři.

- `git pull`

  Do adresáře s lokálním repozitářem stáhne aktuální verzi aktuální větve. Pokud máte lokálně rozpracované změny, stažení se nepovede. Pokud máte uložené změny (commit), git se automaticky pokusí vaše změny sloučit se změnami v globálním repozitáři (merge).

- `git fetch`

  Stáhne celý vzdálený repozitář (všechny větve).

- `git push origin master`

  Do vzdáleného repozitáře `origin` zapíše vaši větev `master`. Vzdálený repozitář může být nastaven tak, že vaše změny musí ještě někdo schválit.

- `git push`

  Zkrácený zápis, pokud jste dříve nastavili pomocí příkazu `git push -set-upstream origin master` název lokální větve a příslušného vzdáleného repozitáře.

Další informace najdete například v dokumentaci.

Úkol 2.1: *Stáhněte si do svých počítačů (naklonujte si) z GitHubu repozitář s poznámkami k tomuto cvičení. V budoucnu si pomocí příkazu* `git pull` *stahujte aktuální verze. Můžete si vytvořit pracovní větev poznámek a v ní si s kódy hrát. V hlavní větvi* `master` *se vám uchová originální verze ze vzdáleného repozitáře.*

Úkol 2.2: *Vytvořte si účet na GitHubu, vytvořte prázdný projekt a navažte si ho s dříve na cvičení vytvořeným lokálním repozitářem pomocí příkazů* `git remote add` *(plný příkaz výše). Následně si do vzdáleného repozitáře nahrajte lokální repozitář pomocí příkazu* `git push`*.*

2.2 .gitignore

Překladače programovacích jazyků často vytvářejí v adresáři vašeho projektu dočasné pomocné soubory, které nechcete, aby se staly součástí repozitáře (tyto soubory nenesou žádnou relevantní informaci, navíc mohou na různých počítačích vypadat jinak podle toho, jaký překladač či jaké vývojové prostředí zrovna použijete). Abyste mohli používat příkazy pro hromadné sledování či zapisování souborů `git add *` a `git commit -a`, musíte GITu naznačit, jaké soubory má ignorovat. K tomu slouží soubor `.gitignore`.

Každé pravidlo v souboru `.gitignore` zabírá jeden řádek. Řádek, který začíná znakem #, je ignorován a může sloužit například jako komentář. Příklady jednotlivých řádků:

- `tajne.txt`

  Ignoruje soubor s názvem `tajne.txt` (může obsahovat třeba přihlašovací údaje k nějaké službě a ty rozhodně nechceme sdílet ani archivovat; nezapomeňte, že co je jednou zapsané v repozitáři, z něj až na výjimky nelze odstranit).

- `*.log`

  Ignoruje všechny soubory s příponou `log`,

- `!important.log`

  ale neignoruje soubor `important.log`.

- `*.[oa]`

  Ignoruje všechny soubory s příponou `o` nebo `a`.

- `temp/`

  Ignoruje všechny soubory v podadresáři `temp`.

- `doc/**/*.pdf`

  Ignoruje všechny soubory s příponou `pdf` v podadresáři `doc` a ve všech jeho podadresářích. Neignoruje však soubory s příponou `pdf` v hlavním adresáři projektu.

Další příklady jsou například zde.

Pokud na GitHubu zakládáte nový projekt, můžete upřesnit, jaký programovací jazyk budete používat a GitHub automaticky vytvoří optimální soubor `.gitignore`.

Úkol 2.3: *Podívejte se do souboru* `.gitignore` *v repozitáři k těmto zápiskům. Zatímco Python si téměř žádné pomocné soubory nevytváří, LaTeX jich generuje požehnaně. Proto je tento soubor celkem dlouhý.*

3 Náhodná procházka

Náhodná procházka je jeden ze základních prostředků, jak simulovat velké množství nejen fyzikálních procesů (například pohyb Brownovské částice, fluktuace akciového trhu, pohyb opilce z hospody atd.). V dalších cvičeních si ukážeme, jak se pomocí náhodné procházky dá jednoduše hledat minimum funkcí (a to i funkcí více proměnných).

Algoritmus pro náhodnou procházku je následující: v každém časovém kroku uděláme krok v d-rozměrném prostoru $\boldsymbol{y}_i \to \boldsymbol{y}_{i+1}$ takovým způsobem, aby pravděpodobnost pohybu do všech směrů byla stejná. Délka kroku l se volí buď náhodná, nebo konstantní.

3.1 Pseudonáhodná čísla

V Pythonu lze pseudonáhodná čísla generovat pomocí několika knihoven.

- Pro základní použití se používá knihovna `random` (dokumentace). Z ní nejdůležitější funkce jsou tyto:
  - `random()`: *reálné* pseudonáhodné číslo x rovnoměrně z intervalu $x \in \langle 0; 1)$.
  - `uniform(a,b)`: *reálné* pseudonáhodné číslo x rovnoměrně z intervalu $x \in \langle a; b \rangle$.
  - `gauss(mu,sigma)`: pseudonáhodné číslo z Gaussovského rozdělení se střední hodnotou `mu` a směrodatnou odchylkou `sigma`.
  - `randint(a,b)`: *celé* pseudonáhodné číslo d z intervalu $a \leq d < b$.
  - `seed(s)`: nastaví počáteční násadu generátoru podle parametru `s` (pro jednu konkrétní násadu bude generátor dávat stejnou sekvenci čísel). Parametr může být jakéhokoliv typu, tedy číslo, řetězec atd. Pokud se parametr neuvede, použije se jako násada systémový čas.
  - `choice(l)`: vybere pseudonáhodně element ze seznamu `l`.
  - `shuffle(l)`: promíchá elementy v seznamu `l`.
- Pro pokročilejší použití je výhodnější modul `numpy.random` (dokumentace). Ta umožňuje zvolit vlastní generátor pseudonáhodných čísel, generovat čísla z celé řady statistických rozdělení a generovat naráz celé vektory či matice.
  - `generator = default_rng()`: Inicializuje standardní generátor pseudonáhodných čísel.
  - `generator = Generator(PCG64())`: Inicializuje specifický generátor pseudonáhodných čísel (v tomto případě PCG-64, což je O'Neillův permutační kongruenční generátor).
  - `generator.random(size=10)`: vektor délky 10 s elementy z rovnoměrného rozdělení z intervalu $\langle 0; 1)$.
  - `generator.normal(size=10)`: vektor délky 10 s elementy z normálního Gaussova rozdělení se střední hodnotou 0 a směrodatnou odchylkou 1.
  - `generator.normal(loc=1, scale=2, size=(10,10))`: matice rozměru 10×10 s elementy z normálního Gaussova rozdělení se střední hodnotou 1 a směrodatnou odchylkou 2.

Úkol 3.1: *Naprogramujte náhodnou procházku ve 2D rovině. Délku kroku volte konstantní, například $l = 1$, směr volte náhodně. Začněte například z bodu $(0; 0)$ a procházku ukončete po N krocích. Případně ji můžete ukončit poté, co se dostanete z předem zadané oblasti, například ze čtverce o hraně délky 100. Uchovávejte celou procházku v poli či seznamu. Nakonec trajektorii vykreslete do grafu.*

Řešení 3.1: *Řešení je naimplementováno v souboru* `RandomWalk2D.py`.

- `RandomDirection2D` *vrátí náhodný směr ve 2D rovině [generuje náhodný úhel ϕ, směr je dán jednotkovým vektorem se složkami $(\cos\phi, \sin\phi)$].*

- `RandomWalk2D` *vykreslí do grafu náhodnou procházku s* `numSteps` *kroky omezenou ve čtverci rozměru* 2 `boxSize`×2 `boxSize` *a náhodnou procházku vrátí.*

- `RandomWalk2DInteractive` *generuje náhodnou procházku a vykresluje ji do grafu krok po kroku. Musí být zapnutý interaktivní mód vykreslování* `plt.ion()` *a v prostředí Spyder vypnuto použití inline grafů příkazem* `%matplotlib auto` *v konzoli REPL.*

Vzorový kód je napsán takovým způsobem, aby mohl být přímočaře rozšířen pro vícerozměrnou náhodnou procházku. Důležité body kódu jsou tyto:

- *Cyklus* `for` *v Pythonu prochází jakýkoliv objekt nazvaný* **iterátor**. *Už jsme se seznámili s cyklem přes prvky seznamu, pole či řádky textového souboru. Pokud chceme jednoduchý cyklus přes po sobě jdoucí celá čísla, použijeme iterátor* `range(start,stop[,step])` *(iteruje se přes celá čísla počínající* `start` *a končící posledním číslem ostře menším než* `stop`*).*

- `numpy.allclose(a,b)` *porovnává prvek po prvku řad* `a` *a* `b` *a pokud jsou všechny prvky blízko sebe v rámci zadané tolerance, vrátí* `True`. *Konečná tolerance je důležitá proto, aby byly ošetřeny případy, kdy důsledkem konečné strojové přesnosti některá fakticky stejná čísla běžné porovnání vyhodnotí jako rozdílná, například* `x = 101*0.1` *a* `y = 10.1`. *Toleranci pro porovnávání lze nastavit.*

  Pro porovnání jednotlivých čísel v mezích tolerance slouží funkce `numpy.isclose(x,y)`.

- *Operátor* `%`*: zbytek po dělení (modulo).*

- *Rozmyslete si dobře podmínku, která zjišťuje, zda jsme uvnitř omezujícího čtverce. Podmínka by dobře posloužila i v případě, kdybychom chtěli implementovat cyklické okrajové podmínky.*

Úkol 3.2: *Zamyslete se nad tím, jak byste realizovali náhodnou procházku s konstantní délkou kroku v $d > 2$ rozměrech. Důležité je dodržet požadavek, aby pohyb do jakéhokoliv směru nastával se stejnou pravděpodobností (esence úlohy tedy spočívá v generování náhodného směru v d-rozměrném prostoru).*

Řešení 3.2: *Zatímco pro směr ve 2D rovině stačí náhodě generovat jeden úhel (předchozí úloha), vícerozměrné úlohy jsou komplikovanější. Přímé rozšíření 2D případu do 3D (či do vyšších dimenzí) za generování více úhlů a použití (hyper-)sférických souřadnic k cíli nevede — takto otrocky generované směry upřednostňují okolí pólů před rovníkem (rozmyslete). K úspěšnému generování náhodného kroku je nutné využít jeden z následujících algoritmů:*

1. *Hyperkoule vepsaná v hyperkrychli.*

   (a) *Nagenerujeme bod v hyperkrychli o hraně délky 2, tj. generujeme vektor $\boldsymbol{v}$ s d složkami, přičemž každá složka je náhodné číslo z rovnoměrného rozdělení $(-1, 1)$.*

   (b) *Zkontrolujeme, zda bod leží uvnitř vepsané jednotkové koule například tak, že spočítáme jeho normu $v = |\boldsymbol{v}|$ a porovnáme, zda $n \leq 1$.*

   (c) *Pokud ne, opakujeme postup od začátku. Pokud ano, nagenerovaný bod promítneme na jednotkovou kouli (jinými slovy vektor $\boldsymbol{v}$ nanormujeme) a získaný jednotkový vektor $\hat{\boldsymbol{v}} \equiv \boldsymbol{v}/v$ udává hledaný směr.*

   Tato metoda je obrovsky neefektivní, pokud je dimenze d vysoká, poněvadž v tom případě většina nagenerovaných bodů leží vně vepsané hyperkoule a je zahozena. Poměr celkového počtu nagenerovaných bodů ku úspěšným zásahům vnitřku hyperkoule lze snadno spočítat. Objem hyperkrychle o hraně délky 2 je

$$V_d^{(krychle)} = 2^d, \tag{27}$$

objem vepsané hyperkoule o poloměru 1 *je*

$$V_d^{(koule)} = \frac{\pi^{\frac{d}{2}}}{\Gamma\left(\frac{d}{2}+1\right)}, \tag{28}$$

kde Γ *je Eulerova gama funkce. Vzájemný poměr*

$$\eta_d \equiv \frac{V_d^{(krychle)}}{V_d^{(koule)}} = \left(\frac{2}{\sqrt{\pi}}\right)^d \Gamma\left(\frac{d}{2}+1\right) \tag{29}$$

udává, kolik bodů musíme průměrně nagenerovat, abychom se trefili do hyperkoule (reciproká hodnota $1/\eta_d$ *určuje pravděpodobnost, že se do hyperkoule trefíme). Zatímco pro* $d=3$ *je* $\eta_3 \approx 1.91$*, pro* $d=10$ *již* $\eta_{10} \approx 401$*, tj. pro nalezení jednoho náhodného směru v desetirozměrném prostoru musíme nagenerovat v průměru přes* 4000 *náhodných čísel. Z posledního vztahu je vidět, že s rostoucí dimenzí roste* η_d *exponenciálně.*

Z tohoto výpočtu také vyplývá, že pokud bychom nezahazovali body ležící mimo hyperkouli, pak bychom ve výsledné procházce výrazně upřednostňovali pohyb podél diagonál, a to tím více, čím vyšší je dimenzionalita procházky (u $d=10$ *bychom podél diagonál vyrazili s více než* 99% *pravděpodobností).*

2. *Náhodný Gaussovský vektor.*

   (a) *Nagenerujeme vektor* $\boldsymbol{n}$ *s* d *složkami, přičemž každá složka je číslo z normálního Gaussovského rozdělení* $N(0,1)$*.*

   (b) *Vektor nanormujeme a získáme hledaný náhodný směr* $\hat{\boldsymbol{n}} \equiv \boldsymbol{n}/n$*.*

   Tato metoda je mnohem přímočařejší než předchozí, předpokladem je jen mít k dispozici generátor čísel vybraných z normálního rozdělení.

   Důkaz, že tato metoda dává opravdu náhodný směr v d *dimenzích, a další informace o metodě se naleznete v článcích [1, 2].*

3. *Speciální případ* $d=3$ *(náhodný let).*

   (a) *Generujeme dvě náhodná čísla* $\xi_{1,2}$ *z rovnoměrného rozdělení na intervalu* $\langle 0;1)$*.*

   (b) *Sférické úhly jednotkového směru jsou pak*

   $$\begin{aligned} \phi &= 2\pi\xi_1, \\ \theta &= \arccos\left(1-2\xi_2\right), \end{aligned} \tag{30}$$

   takže hledaný jednotkový vektor $\hat{\boldsymbol{n}}$ *do náhodného směru má komponenty*

   $$\begin{aligned} \hat{n}_x &= \sin\theta\cos\phi = \sqrt{1-\left(1-2\xi_2\right)^2}\cos 2\pi\xi_1, \\ \hat{n}_y &= \sin\theta\sin\phi = \sqrt{1-\left(1-2\xi_2\right)^2}\sin 2\pi\xi_1, \\ \hat{n}_z &= \cos\theta = 1-2\xi_2. \end{aligned} \tag{31}$$

   Ve více rozměrech je tento přístup prakticky nerealizovatelný (vede na problém inverzních funkcí k funkcím daným řadou goniometrických funkcí).

Náhodná procházka v d*-rozměrném prostoru je naprogramována v souboru* `RandomWalk.py`*. Funkce* `RandomDirection` *generuje směr pomocí 1. metody, funkce* `RandomDirectionGaussian` *pomocí 2. metody. Zkuste si spočítat náhodnou procházku pro* $d=10$ *oběma metodami. Uvidíte, že i pro takto relativně „malou“ dimenzi je rozdíl ve výpočetních časech je dramatický.*

4 Hledání minima funkce

Náhodnou procházku lze úspěšně použít k hledání minima funkce obecně více proměnných. Představte si funkci dvou proměnných jako zvlněnou „krajinu" v noci. Potřebujete se vrátit k chatě, která se nachází pod vámi hluboko v úkolí. Je tma a nevidíte jakým směrem se vydat. Zkusíte tedy udělat náhodný krok a pokud povede dolů, vykročíte. Pokud by však krok vedl nahoru, zůstanete na místě a zkusíte nový směr.

Úkol 4.1: *Rozšiřte program pro náhodnou procházku tak, aby hledal minimum funkce dvou proměnných $f(x, y)$. Otestujte svůj program pro kvadratickou funkci*

$$f(x, y) = x^2 + y^2 \tag{32}$$

a pro Rosenbrockovu funkci

$$g(x, y) = (a - x)^2 + b\left(y - x^2\right)^2 \tag{33}$$

vypadající jako velmi pozvolna klesající hluboké údolí ve tvaru paraboly. Tato funkce se používá k testování rychlosti a efektivity minimalizačních algoritmů. Její minimum se nachází v bodě (a, a^2) a hodnoty parametrů nejčastěji se volí $a = 1, b = 100$.

Implementujte vhodným způsobem ukončení náhodné procházky, tj. okamžik, kdy jste již dorazili do minima funkce.

Řešení 4.1: *Vzorový kód je naimplementován v souboru* `RandomWalkMinimize.py` *a vychází z náhodné vícerozměrné procházky z modulu* `RandomWalk.py` *(z tohoto modulu kód využívá funkci* `RandomDirectionGaussian`*). K minimalizaci jsou v kódu dvě různé metody:*

- `Minimize`*: Hledá minimum funkce* `Function` *z počátečního bodu daného parametrem* `initialCondition`*. Pokud tento parametr není specifikován, zvolí se počáteční bod náhodně z* `dimension`*-rozměrné hyperkrychle se středem v počátku a délkou hrany 2∗*`initialConditionBox`*. Výpočet je ukončen, pokud se kódu* `maxFailedSteps`*-krát po sobě nepodaří udělat úspěšný krok (krok směrem k menší funkční hodnotě). Každý náhodný krok má konstantní délku danou parametrem* `stepSize`*.*

- `MinimizeAdaptive`*: Předchozí metoda hledání minima má chybu $\Delta x_i \approx$* `stepSize`*. Pro zmenšení chyby je v ní nutné zmenšit délku kroku náhodné procházky. Pokud tak učiníme, výpočetní čas T se výrazně prodlouží (desetkrát za každý řád zpřesnění výsledku, tj. $T \propto 1/$*`stepSize`*). Mnohem vhodnější je začít s velkým krokem a krok zmenšovat postupně. Tak postupuje tato funkce: začne s krokem* `initialStepSize` *a pokaždé, když se jí nepodaří* `maxFailedSteps`*-krát provést úspěšný krok, zmenší délku kroku na polovinu. Výpočet probíhá do chvíle, dokud je délka kroku větší než* `finalStepSize`*. Konečná chyba je tedy $\Delta x_i \approx$* `finalStepSize` *a doba výpočtu roste pouze logaritmicky se zmenšováním této chyby, tj. $T \propto 1/\log$* `finalStepSize`*.*

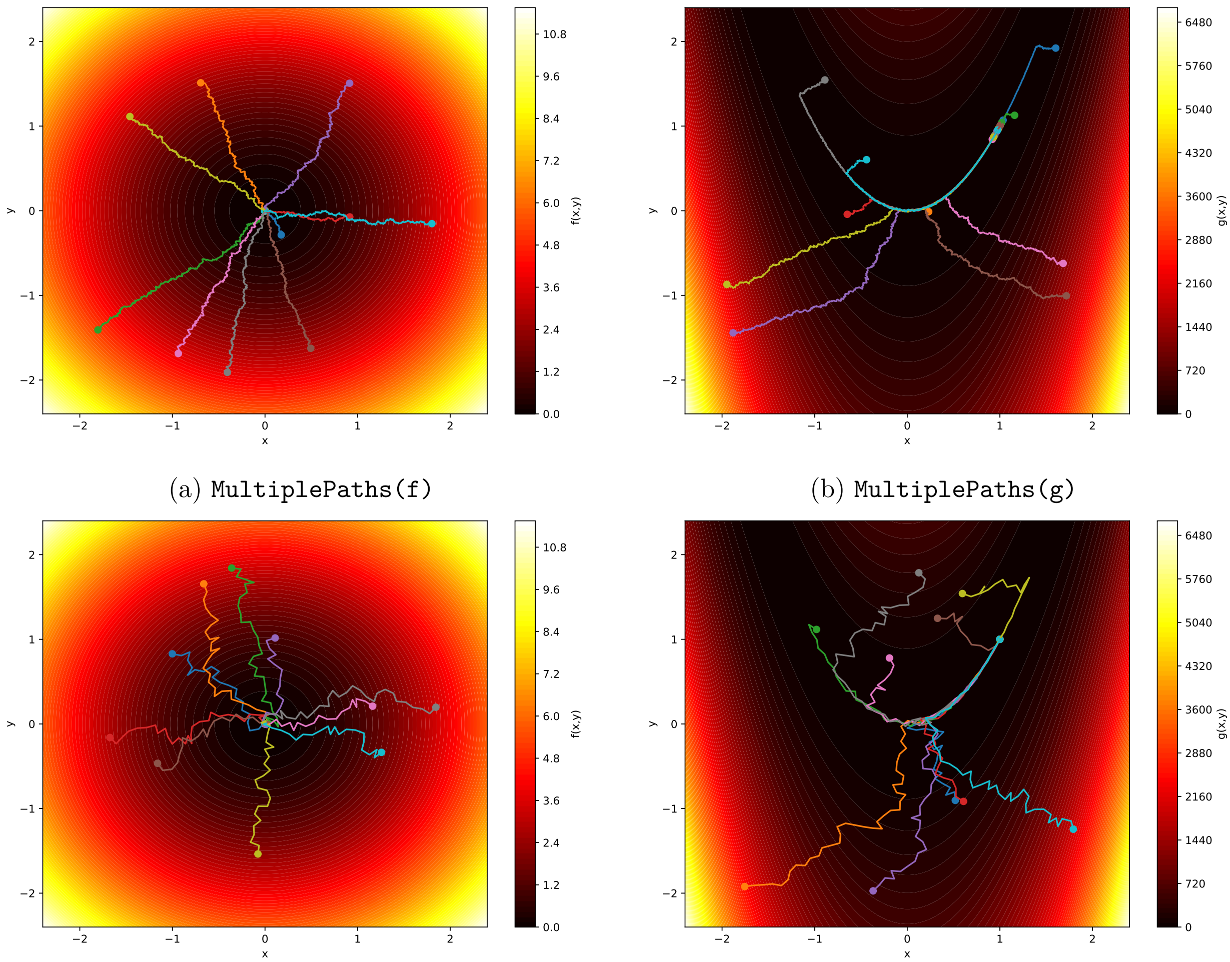
Obě funkce vracejí řadu s celou náhodnou procházkou. Nalezené minimum je tedy v posledním bodě této řady.

V kódu se vyskytuje jedna nová věc, a to je metoda `format` *řetězce. Tato metoda postupně na místa řetězce vyznačená dvojicí znaků {} vloží své argumenty, přičemž argumenty mohou být libovolné objekty. Pro více informací k formátování řetězců doporučuji tento výborný návod.*

Konkrétní funkce $f(x, y)$ a $g(x, y)$ ze zadání jsou v souboru `Minimize.py`*.*

Úkol 4.2: *Náhodnou procházku zakreslete jako čáru do grafu společně s konturovým grafem potenciálu. Návod na nakreslení konturového grafu pomocí funkce* `matplotlib.pyplot.contourf` *naleznete v souboru* `Contourf.py`*.*

Řešení 4.2: *Vzorový kód je v souboru* `Minimize.py` *a využívá jako modul* `RandomWalkMinimize.py`*. Obsahuje následující funkce:*



(a) `MultiplePaths(f)` (b) `MultiplePaths(g)`

(c) `MultiplePathsAdaptive(f)` (d) `MultiplePathsAdaptive(g)`

Obrázek 5: Minimalizace kvadratické funkce (32) a Rosenbrockovy funkce (33) pomocí náhodné procházky a deseti náhodných trajektorií. *1. řádek:* Minimalizace metodou `Minimize` s délkou kroku `stepSize` $= 0.01$ a kritériem ukončení `maxFailedSteps` $= 100$. Chyba určení minima je $\Delta x, y \approx 0.01$, což v případě Rosenbrockova minima dává velkou chybu: Rosenbrockovo „údolí“ je velmi pozvolné podél a strmé napříč, což způsobuje, že trajektorie končí v širokém okolí skutečného minima v bodě $(x, y) = (1, 1)$. Počet kroků, než je výpočet ukončen, je zde ≈ 1000. *2. řádek:* Minimalizace vylepšeným algoritmem s postupně se zmenšující krokem. Na počátku je krok velký, `initialStepSize` $= 0.1$, ale postupně se zmešuje až k `finalStepSize` $= 10^{-6}$. To stačí i pro přesné nalezení minima Rosenbrockovy funkce. Počet nutných kroků zde je ≈ 3000.

- `ShowGraph`: *Vykreslí konturový graf funkce* `Function` *a do něj zakreslí křivky (náhodné procházky) z parametru* `paths`. *Pokud tento parametr není specifikován, vykreslí pouze konturový graf s funkcí. Meze funkce pro vykreslení jsou dány parametrem* `boxSize`.
- `MultiplePaths`: *Vypočítá celkem* `numPaths` *jednoduchých náhodných minimalizačních procházek a vykreslí je v grafu společně s konturovým grafem minimalizované funkce.*
- `MultiplePathsAdaptive`: *Vykreslí totéž, avšak použije metodu s adaptivním krokem.*

Výsledky pro funkce f a g ze zadání jsou zobrazeny na obrázku 5.
V kódu jsou následující vychytávky:

- `Function.__name__` *vrátí originální název funkce. Každá funkce v Pythonu má hodnotu tohoto atributu nastavenou.*
- *V cyklu* `for _ in range(0, numPaths)` *je proměnná označena podtržítkem* _. *To je v Pythonu běžně užívané označení pro proměnnou, jejíž hodnota se nikde nepoužívá.*
- *Z argumentů funkce* `MultiplePaths` *chceme většinu pojmenovaných argumentů předat přímo funkci* `Minimize`, *aniž by nás zajímalo, kolik takových argumentů je a jak se jmenují. K tomu slouží proměnná uvozená dvěma hvězdičkami* `**kwargs`[2]. *Ta v sobě nese všechny nepoužité pojmenované argumenty (nikoliv tedy* `initialConditionBox`, *který musíme při volání funkce* `Minimize` *předat explicitně). Typ proměnné* `kwargs` *je slovník.*

Úkol 4.3: *Rozšiřte kód tak, aby počítal i vícerozměrnou náhodnou procházku, a najděte pomocí něho minimum funkce čtyř proměnných*

$$\begin{aligned} h(s,t,u,v) = &\frac{1}{4}\left(s^2+t^2+u^2+v^2\right) \\ &-\frac{1}{2}\left[\left(s^2+t^2\right)\left(2-s^2-t^2-u^2-v^2\right)+(su-tv)^2\right] \\ &+\frac{s}{2}\sqrt{2-s^2-t^2-u^2-v^2}. \end{aligned} \tag{34}$$

Řešení 4.3: *Volání funkce* `MinimizeAdaptive(h, dimension=4, initialConditionBox=1)` *vypočítá minimum v bodě*

$$\begin{aligned} s_{min} &\approx -0.913 \\ t_{min} = u_{min} = v_{min} &\approx 0.000 \\ h_{min} \equiv h(s_{min}, t_{min}, u_{min}, v_{min}) &\approx -0.771. \end{aligned} \tag{35}$$

V případě minimalizace této funkce je nutné ošetřit odmocninu, pod kterou se může během náhodné procházky objevit záporné číslo, což způsobí konec výpočtu s chybou. V kódu je to vyřešeno tak, že v případě kroku do této „nedovolené oblasti" vrátí funkce `h` *hodnotu ∞ (v Pythonu* `float("inf")`*), která je větší než všechna možná jiná čísla, a tím tento krok zakáže. Ze stejného důvodu je zmenšena oblast hledání počáteční polohy pomocí parametru* `initialConditionBox=1`.

Jednoduchá náhodná procházka funguje dobře pro funkce s jedním minimem. V obecném případě má však funkce více lokálních minim a právě uvedený algoritmus skončí náhodně v jednom z nich, ze kterého se již nedokáže dostat ven. Při tom rozhodně nemusí jít o minimum nejhlubší (globální).

Hledání globálního minima funkce mnoha proměnných je obecně velmi komplexní problém. Dva nejjednodušší postupy, kterými můžeme vylepšit stávající metodu pomocí náhodné procházky, jsou následující:

[2] kw je zkratka pro *keywored*; proměnná se samozřejmě může jmenovat jakkoliv, `kwargs` je jen standardně používané označení.

- Provedeme několik náhodných procházek, které obecně dojdou do různých lokálních minim. Následně porovnáme konečné funkční hodnoty a vybereme to minimum, které má hodnotu nejnižší.

- Provedeme jednu náhodnou procházku doplněnou o *Metropolisův algoritmus*.

4.1 Metropolisův algoritmus

Metropolisův algoritmus rozšiřuje náhodnou procházku o konečnou teplotu. Je inspirován termodynamickým Boltzmannových rozdělením energie: máme tepelnou energii, díky které můžeme při náhodné procházce s určitou pravděpodobností udělat krok i „do kopce“, avšak čím je kopec strmější, tím bude pravděpodobnost takového kroku menší.

Předpokládejme, že jsme na vrstevnici s funkční hodnotou f a nová funkční hodnota po provedení kroku náhodné procházky by byla $f_{\text{nová}} > f$. Při minimalizaci pomocí obyčejné náhodné procházky bychom tento krok neprovedli. V Metropolisově algoritmu krok provedeme s pravděpodobností

$$p = \mathrm{e}^{\frac{f - f_{\text{nová}}}{T}}, \tag{36}$$

kde T je parametr, který má roli „teploty“: pokud $T = 0$, žádný tepelný pohyb neexistuje, krok do kopce nikdy neprovedeme a vracíme se tak k obyčejné minimalizaci. Pokud $T \to \infty$, uděláme krok do kopce s pravděpodobností $p \to 1$, což znamená, že tepelný pohyb zcela převládá, my se pohybujeme zcela náhodně a potenciál pod sebou vůbec necítíme.

V praxi je největší umění zvolit správnou hodnotu teploty. Pokud zvolíme teplotu nízkou, skončíme v lokálním minimu a už se z něj nedostaneme, pokud naopak příliš vysokou, budeme chaoticky procházet krajinou naší funkce a žádné minimum nenajdeme. Dobrá volba je začít spíš s vyšší teplotou a teplotu postupně snižovat. Jakmile se ocitneme zaseklí v nějakém minimu, můžeme teplotu zase trochu zvýšit a tím vyzkoušet, zda se nepřesuneme do nějakého minima hlubšího.

Úkol 4.4: *Naprogramujte Metropolisův algoritmus a odlaďte ho na případu funkce*

$$r(x, y) = x^4 - 2x^2 + x + y^2. \tag{37}$$

Tato funkce má dvě lokální minima (jedná se o vzorovou funkci ze souboru `Contourf.py`*).*

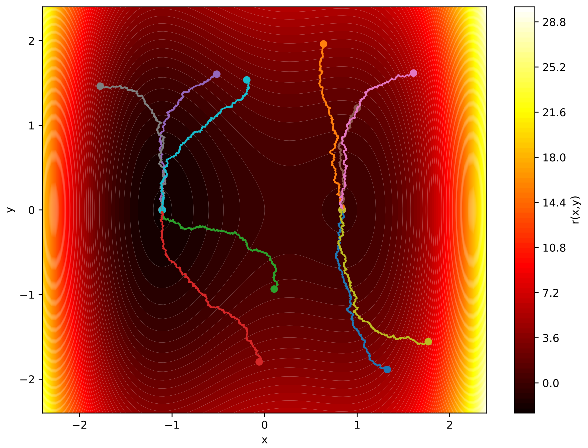
Řešení 4.4: *Pro ukázku, že standardní minimalizace pomocí náhodné procházky vede náhodně do různých lokálních minim, slouží obrázek 6(a).*

Metropolisův algoritmus je naprogramován v souboru `MetropolisMinimize.py`*.*

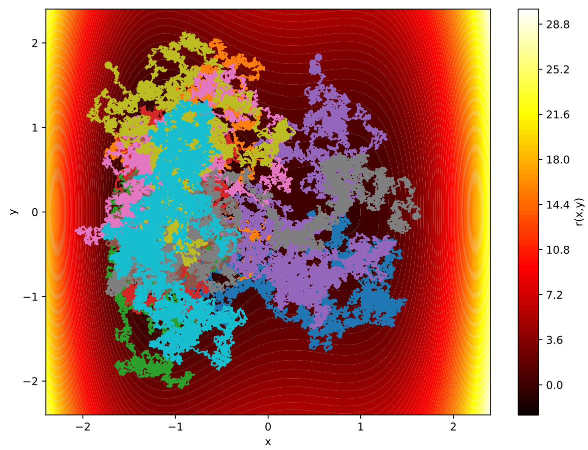
- `Minimize`*: Minimalizuje funkci* `Function` *se zadanou teplotou* `temperature` *a s fixní délkou kroku* `stepSize`*. Výpočet je ukončen přesně po* `maxSteps` *krocích. Kritérium ukončení výpočtu z metody bez teploty zde nebude fungovat, jelikož konečná teplota způsobuje neustálý „tepelný pohyb“, nikdy tedy nedojde k úplnému zastavení, ani pokud dosáhneme minima funkce.*

  Výsledek je zobrazen na obrázku 6(b). Je vidět, že oproti případu bez teploty se dostaneme častěji k okolí globálního minima, avšak důsledkem celkem vysoké teploty nalezená poloha výrazně fluktuuje. Navíc, a to zde není zobrazeno, výsledek velmi záleží na kombinaci teploty a délky kroku.

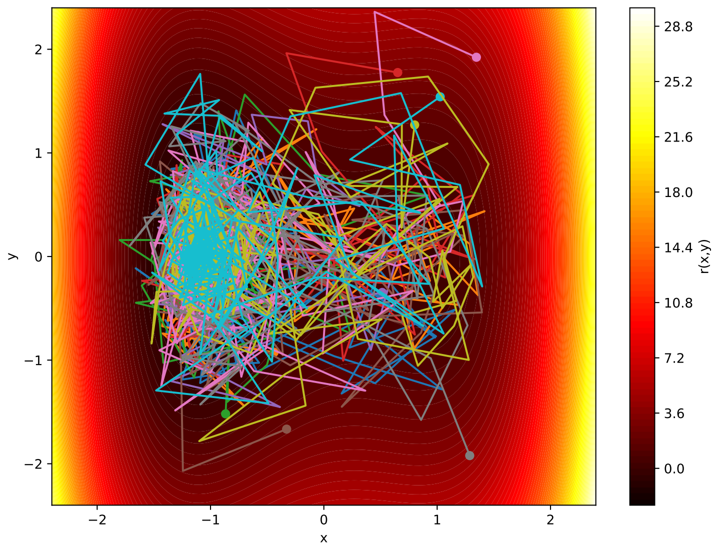
- `MinimizeAdaptive`*: Vylepšení spočívající v postupném zmenšování teploty a zároveň kroku. Výpočet začne s krokem* `initialStepSize` *a teplotou* `initialTemperature` *a po každých* `numStepsChange` *krocích déku kroku i teplotu vydělí dvěma. Výpočet je ukončen po dosažení délky kroku* `finalStepSize`*.*



(a) `MultiplePaths(r)` pro `RandomWalkMinimize`.



(b) `MultiplePaths(r, numSteps=30000)` pro `MetropolisMinimize`.



(c) `MultiplePathsAdaptive(r)` pro `MetropolisMinimize`.

Obrázek 6: Porovnání různých metod minimalizace dvoujámové funkce (37). (a) Minimalizace obyčejnou náhodnou procházkou, která dokonverguje do náhodného lokálního minima. Následně je možné vybrat to minimum, které je hlubší. (b) Metropolisův algoritmus s konstantní teplotou $T = 1$. Kvůli vysoké teplotě dráha výrazně fluktuuje, avšak postupně dokonverguje k okolí hlubšího globálního minima. (c) Metropolisův algoritmus s postupně se zmenšujícím krokem a teplotou. Zde do globálního minima postupně zkonvergují všechny trajektorie. Počet použitých kroků je 2000.

4.2 Minimalizace pomocí knihovny SciPy

Python obsahuje funkci pro hledání minima `minimize` v knihovně `scipy.optimize` (z této knihovny jsme v jednom z prvních cvičení využívali funkci `least_squares` pro hledání optimální hodnoty parametrů funkce fitující zadaná data metodou nejmenších čverců).

Úkol 4.5: *Prostudujte dokumentaci k funkci* `minimize` *a vytvořte kód, který tuto funkci využije k najití minima všech doposud studovaných funkcí dvou a více proměnných.*

Řešení 4.5: *Knihovní funkci voláme příkazem* `minimize(f, initialCondition)`*, kde parametr* `initialCondition` *je n-tice udávající počáteční bod, ze kterého minimalizační procedura vystartuje.*

V případě funkce s více lokálními minimy tato knihovní funkce nenajde nutně minimum globální. Lze však pomocí volitelného parametru `method` *vybrat metodu minimalizace, která bude úspěšnější. Pokročilá varianta Metropolisova algoritmu je v v knihovně* `scipy.optimize` *naprogramována pod názvem* `basinhopping` *(dokumentace).*

4.3 Shrnutí

- Jeden z nejjednodušších algoritmů na hledání minima (maxima) funkce je pomocí náhodné procházky. K její implementaci stačí mít generátor náhodných čísel.

- Úspěch algoritmů založených na náhodné procházce je podmíněn tím, že každý krok procházky musí vést do libovolného směru se stejnou pravděpodobností. Při procházce v rovině toho lze docílit náhodným generováním úhlu. Ve vícerozměrné procházce je nejvýhodnější použít algoritmus založený na Gaussovském náhodném vektoru (za předpokladu, že umíme generovat čísla z Gaussovského rozdělení; jeden jednoduchý takový generátor bude obsahem dalších cvičení).

- Obecná funkce může mít více lokálních minim, přičemž náhodná procházka nás zavede do jednoho z nich, které nemusí být nejhlubší (globální). K nalezení globálního minima lze využít Metropolisův algoritmus, který k náhodné procházce přidá tepelný pohyb, a tím umožní „vyskočit“ z mělkého lokálního minima.

  Metropolisův algoritmus se využívá i pro jiné termodynamické úlohy, například k modelování spinových systémů při konečné teplotě, čímž lze studovat fázový přechod feromagnet ↔ paramagnet.

5 LaTeX

Nedílnou součástí vědecké práce je prezentování výsledků, přičemž chceme, aby dokument bylo jednoduché napsat, a zároveň aby dobře vypadal. Jelikož fyzikální odborné texty obsahují velké množství rovnic a symbolů, není nejvhodnější volbou používat textové editory pro kancelářskou práci, jejichž možnosti psaní rovnic a použití vědeckých stylů jsou celkem omezené. Ve fyzice je nejběžnější psát texty v systému LaTeX.

Dnes se jedná o nesmírně obsáhlý balík různých knihoven a doplňků, pomocí kterého lze

- psát dobře vypadající a typograficky správné odborné texty s matematickými rovnicemi, obrázky a tabulkami,

- vybírat z množství profesionálně připravených stylů obsahujících podporu tvorby obsahu, rejstříku, poznámek pod čarou, kapitol, seznamů, referencí a dalších vychytávek,

- stáhnout si přímo styl časopisu či knihy, pro kterou text připravujeme,

- používat tisíce matematických symbolů či druhů písem (lze psát třeba ve švabachu, gotickém písmu či elfím písmu, jste-li fanoušci díla J.R.R. Tolkiena),

- využívat různá makra a doplňky k jednoduchému vytváření speciálních objektů (například chemických vzorců nebo Feynmannových diagramů),
- vytvářet robustní prezentace (například pomocí knihovny Beamer),
- nebo sázet notové materiály a kdovíco dalšího (rajčata však zatím ne).

Systém LaTeX má dvě vrstvy:

1. TeX: sazeč (zajišťuje, aby vše bylo na stránce tam, kde má být),
2. LaTeX: typograf (zajišťuje, aby dokument dobře vypadal).

Program TeX začal vznikat v 70. letech 20. století a byl určen sázení textu a matematických rovnic při zachování vysoké typografické úrovně výsledného dokumentu. LaTeX je pak nadstavba maker, která psaní dokumentů velmi zjednodušuje a zcela zakrývá sazečskou práci. Příkazy v TeXu jsou však stále dostupné.

Dokument napsaný v LaTeXu striktně odděluje *text* (obsah) a *styl* (vzhled), přičemž my využijeme standardní LaTeXovský styl a budeme se zabývat výhradně tím, jak napsat text. Jeho psaní se podobá programování: zdrojový soubor je textový dokument (nebo soubor textových dokumentů), který sestává z našeho textu a doplňujících příkazů. Chceme-li vidět, jak bude výsledný dokument vypadat, musíme soubor „přeložit“.

Existují různé *distribuce* LaTeXu. Jedna z nejpoužívanějších je MiKTeX. Dále je nutné mít k dispozici textový editor, přičemž vhodný je ten, který rozumí TeXovským příkazům a bude umět zdrojový text poslat k překladu a zobrazit. Pokročilé editory umějí navigovat mezi zdrojovým textem k přeloženému výsledku (většinou do PDF souboru) a naopak. Z volně dostupných editorů jsou nejčastěji používané[3]

- **TeXworks**: Editor se základními funkcemi. Je součástí distribuce MiKTeX.
- TeXnicCenter: Pokročilejší editor, podpora použití projektů.
- TeXstudio: Pokročilý editor, podpora projektů, automatického doplňování, jednoduchá práce s obrázky, možnost tvorby záložek, pomocníci po tvorbu rovnic a tabulek, zvýrazňování syntaktických chyb.
- Visual Studio Code s pluginem LaTeX Workshop: Méně funkcí co se týče samotného LaTeXu a komplikovaná instalace, avšak plná podpora všech funkcí tohoto rozšířeného vývojového prostředí (integrace s Gitem, pokročilé vyhledávání, programátorské možnosti editace).
- Overleaf: Online editor, ideální pro práci v týmu. Nemusíte nic isntalovat, potřebujete však připojení k internetu. Komplikované může být rozchodit nějaké méně rozšířené balíčky.

Úkol 5.1: *Nainstalujte si na svůj počítač nějakou distribuci LaTeXu (doporučuji MiKTeX) a editor (doporučuji TeXstudio nebo TeXnicCenter).*

5.1 Formát TeXovského souboru

- ASCII nebo UTF-8 kódování (lze použít i obskurnější kódování, ale v dnešní době je UTF-8 dostatečně univerzální).
- *Mezery*: více mezer je interpretováno jako jedna mezera, jednoduché zalomení řádku také jako jedna mezera.
- *Nový odstavec*: dvě zalomení řádku po sobě.

[3]Kromě prvního editoru mám zkušenost se všemi ostatními. Tento text píšu ve Visual Studiu Code, avšak pro začátek doporučuji nejlépe TeXstudio nebo TeXnicCenter.

- *Příkazy*: uvozeny znakem `\`, jejich povinné parametry ve složených závorkách `{...}`, volitelné parametry v hranatých závorkách `[...]`. U jmen příkazů záleží na velikosti písmen.
- *Rovnice v textu*: oddělena znaky `$...$`. Některé příkazy lze použít pouze uvnitř rovnic.
- *Komentář*: uvozen znakem `%`. Vše za tímto znakem až do konce řádky je ingnorováno.

5.2 Struktura TEXovského souboru

5.2.1 Preambule

Jedná se o první řádky souboru, než začne vlastní tělo dokumentu. Obsahuje výčet všech použitých balíčků a parametry, které se použijí pro styl textu.

- `\documentclass[a4paper,twoside,11pt,twocolumn]{article}`: Základní specifikace stylu dokumentu (v tomto případě papír velikosti A4, dvoustránkový tisk, základní velikost písma 11 bodů, dvousloupcová sazba). Tento příkaz je povinný, musí být vždy přítomen.
- `\usepackage{epsfig}`: Použije se balíček `epsfig`.
- `\def\abs#1{\left|#1\right|}`: Definuje makro `\abs` s jedním parametrem (absolutní hodnota).

Nejčastěji používané balíčky jsou tyto:

- `\usepackage{amsfonts,amsmath,amssymb}`:[4] Rozšiřuje množství použitelných písem, matematických symbolů a struktur (např. snazší psaní matic, víceřádkových rovnic atd.).
- `\usepackage[utf8]{inputenc}`: Specifikuje UTF-8 jako vstupní kódování (jinak je očekáváno ASCII).
- `\usepackage[czech]{babel}`: Udávající české formátování (například české uvozovky) a české názvy (například Obsah, Rejstřík).
- `\usepackage{epsfig}`: Umožní vkládání vektorových EPS souborů.
- `\usepackage[unicode]{hyperref}`: Umožní vkládání hypertextových odkazů a učiní klikabilní i odkazy na rovnice, obrázky, stránky či kapitoly v textu pro snazší navigaci.

5.2.2 Tělo dokumentu

Tělo je uvozeno příkazy

```
\begin{document}
    ...
\end{document}
```

a do něj píšeme vlastní text dokumentu. Vše, co se nachází za příkazem `\end{document}`, je ignorováno.

Jednoduchý příklad LATEXovského dokumentu s vysvětlením základů psaní textu je v repozitáři v adresáři LaTeX a jmenuje se **`dokument.tex`**. K jeho přeložení budete potřebovat i přiložený EPS soubor **`kubik.eps`**.

Pokročilejším přikladem je přímo tento soubor (i jeho zdroják je v repozitáři).

Úkol 5.2: *Prostudujte si vzorový LATEXovský soubor* **`dokument.tex`** *a napište v LATEXu vlastní pojednání o nějakém svém oblíbeném vědci, fyzikálním (případně matematickém) teorému či rovnici. Dokument by měl obsahovat aspoň jednu rovnici a nejlépe i obrázek. Textová část stačí na jednu stránku.*

[4] AMS = American Mathematical Society.

5.3 Další návody a odkazy

- Ne příliš stručný úvod do sys1tému LaTeX 2_ε: Vynikající srozumitelný, přehledný a čtivý návod v češtině. Doporučuji prostudovat.
- Jak na LaTeX: Webový seriál, obsahuje i určité pokročilejší vychytávky.
- The comprehensive LaTeX Symbol List: Několikasetstránkový dokument se všemi možnými použitelnými matematickými i nematematickými symboly.
- LaTeX: Asi nejpodrobnější příručka dostupná na webu.

6 Histogram

V tomto cvičení se vrátíme k náhodným číslům, se kterými jsme se již setkali při programování náhodné procházky 3. Nyní se podíváme hlouběji na jejich vlastnosti, naučíme se zobrazit hustotu pravděpodobnosti jejich rozdělení (histogram), vytvoříme triviální generátor čísel z Gaussovského normálního rozdělení a generátor čísel z libovolného rozdělení zadaného hustotou pravděpodobnosti nebo distribuční funkcí.

Histogram je jeden z klíčových objektů v mnoha oblastech fyziky, kde se pracuje s náhodnými veličinami. To je vpodstatě celá kvantová mechanika, a tudíž obory jako je atomová, jaderná, či subjaderná fyzika. S náhodnými veličinami se setkáte samozřejmě také v klasické statistické fyzice, ale také například v meteorologii či dalších oborech. Stojí proto za to se s ním seznámit podrobně.

6.1 Základní definice a tvrzení z teorie pravděpodobnosti

V následujícím textu budeme značit X spojitou náhodnou veličinu[5] s hodnotami v intervalu $x \in \langle a, b\rangle$[6]. Důležité pojmy a vztahy pro nás budou:

- **Hustota pravděpodobnosti** $\rho(x)$**:** Pravděpodobnost, že náhodná veličina X bude nabývat hodnoty z intervalu $\langle x_1, x_2\rangle \subset \langle a, b\rangle$, je
$$\Pr\left[x_1 \leq X \leq x_2\right] = \int_{x_1}^{x_2} \rho(x)\mathrm{d}x. \tag{38}$$
Hustota pravděpodobnosti je normalizovaná na definičním oboru,
$$\int_a^b \rho(x)\mathrm{d}x = 1 \tag{39}$$
(pravděpodobnost, že bude náhodná veličina nabývat libovolné ze svých povolených hodnot, je 1 = jistý jev). Hustotu pravděpodobnosti lze vždy rozšířit na celou množinu $\mathbb{R}$, pokud dodefinujeme $\rho(x) = 0$ pro $x < a$ a $x > b$.
- **Distribuční funkce (kumulovaná hustota pravděpodobnosti)** $F(x)$**:** Neklesající spojitá funkce s oborem hodnot $\langle 0, 1\rangle$ daná integrálem hustoty pravděpodobnosti[7]
$$F(x) = \int_a^x \rho(x')\mathrm{d}x'. \tag{41}$$

[5] V teorii pravděpodobnosti se náhodné veličiny značí obvykle velkým písmenem.

[6] Náhodná veličina X je ve skutečnosti velmi abstraktní objekt. Obecně se definuje na měřitelném prostoru $(\mathcal{X}, \mathcal{A}, \mu)$, kde $\mathcal{X}$ je množina možných hodnot náhodné veličiny X, $\mathcal{A}$ je σ-algebra nad množinou $\mathcal{X}$ (neprázdný systém množin uzavřený na spočetné sjednocení a obsahující prázdnou množinu a množinu $\mathcal{X}$) a μ je míra množiny $\mathcal{M} \subset \mathcal{X}$ (nezáporná σ-aditivní množinová funkce nulová pro prázdnou množinu a jednotková pro celou množinu $\mathcal{X}$). Tato definice v sobě zahrnuje jak náhodné veličiny s diskrétními možnými hodnotami (jako je například hod kostkou), tak náhodné veličiny se spojitými možnými hodnotami, kterým se věnujeme v této sekci.

[7] Nebo obráceně, hustota pravděpodobnosti je derivace distribuční funkce,
$$\rho(x) = \frac{\mathrm{d}F}{\mathrm{d}x}. \tag{40}$$

Platí tedy díky normalizaci (39)

$$F(a) = 0,$$
$$F(b) = 1.$$

Rozšíříme-li obor hodnot náhodné veličiny na všechna reálná čísla stejným způsobem, jako jsme naznačili u hustoty pravděpodobnosti, platí navíc

$$F(x < a) = 0,$$
$$F(x > b) = 1.$$

Pravděpodobnost, že náhodná veličina X bude nabývat hodnoty z intervalu $\langle x_1, x_2 \rangle$, je pak jednoduše

$$\Pr\left[x_1 \leq X \leq x_2\right] = F(x_2) - F(x_1). \tag{42}$$

- **Střední hodnota:**[8]

$$\mathrm{E}\left[X\right] = \int_{-\infty}^{\infty} x\rho(x)\mathrm{d}x. \tag{43}$$

- **Rozptyl:**

$$\sigma_X^2 = \mathrm{E}\left[X^2\right] - \mathrm{E}\left[X\right]^2 = \int_{-\infty}^{\infty} x^2\rho(x)\mathrm{d}x - \left[\int_{\infty}^{\infty} x\rho(x)\mathrm{d}x\right]^2. \tag{44}$$

- **Výběrová střední hodnota:** Pokud máme soubor n hodnot náhodné veličiny X, které označíme $\{x_1, x_2, \ldots, x_n\}$ (výběr), pak výběrová střední hodnota je dána aritmetickým průměrem,

$$\overline{X} = \frac{1}{n}\sum_{j=1}^{n} x_n. \tag{45}$$

Čím mohutnější máme výběr, tím lépe výběrová střední hodnota aproximuje střední hodnotu,

$$\overline{X} \xrightarrow{n\to\infty} \mathrm{E}\left[X\right]. \tag{46}$$

- **Histogram:** Graf (obvykle sloupcový), který aproximuje distribuční funkci náhodné veličiny X na základě hodnot výběru $\mathcal{V} = \{x_j, j = 1, \ldots, n\}$. Graf se skládá z $N \ll n$ intervalů (sloupců) obvykle konstantní šířky pokrývající obor hodnot náhodné veličiny $\langle a, b \rangle$, přičemž výška sloupce na konkrétním intervalu je rovna počtu hodnot z výběru $\mathcal{V}$, které do intervalu padnou. Pokud histogram správně nanormujeme, získáme (poněkud zubatou) aproximaci distribuční funkce.

- **Nezávislé náhodné veličiny:** Dvě náhodné veličiny X a Y jsou nezávislé, pokud jedna neovlivňuje druhou. Sdružená hustota pravděpodobnosti nezávislých náhodných veličin je dána součinem dílčích hustot pravděpodobnosti,

$$\rho_{X,Y}(x, y) = \rho(x)\rho(y). \tag{47}$$

Například věk a výška osoby nejsou nezávislé veličiny (pro děti bude rozdělení jejich výšek jiné než pro dospělé), zatímco věk osoby a její krevní skupina nezávislé veličiny jsou.

- **Součet dvou náhodných veličin:** Pokud máme náhodnou veličinu X s hustotou pravděpodobnosti $\rho_X(x)$ a náhodnou veličinu Y s hustotou pravděpodobnosti $\rho_Y(y)$, přičemž obě náhodné veličiny jsou nezávislé, pak náhodná veličina

$$Z = X + Y \tag{48}$$

[8]**E**xpectation value

bude mít hustotu pravděpodobnosti ρ_Z danou *konvolucí* hustot ρ_X a ρ_Y,

$$\rho_Z(z) = \int_{-\infty}^{\infty} \rho_X(\xi)\rho_Y(z-\xi)\mathrm{d}\xi. \tag{49}$$

Střední hodnota a rozptyl náhodné veličny Z jsou dány součtem

$$\begin{aligned} \mathrm{E}\left[Z\right] &= \mathrm{E}\left[X\right] + \mathrm{E}\left[Y\right], \\ \sigma_Z^2 &= \sigma_X^2 + \sigma_Y^2. \end{aligned} \tag{50}$$

- **Centrální limitní věta:** Je-li náhodná veličina Y daná součtem m vzájemně nezávislých náhodných veličin $X^{(1)}, X^{(2)}, \ldots, X^{(m)}$ se shodným rozdělením s hustotou pravděpodobnosti $\rho(x) = \rho_{X^{(j)}}(x)$, jehož střední hodnota je $\mu \equiv \mathrm{E}\left[X^{(j)}\right] < \infty$ a $\sigma^2 \equiv \sigma^2_{X^{(j)}} < \infty$, $j = 1, \ldots, m$, pak

  $$Y \sim N(m\mu, m\sigma^2), \tag{51}$$

  kde $N(\mu, \sigma^2)$ je Gausovské normální rozdělení se střední hodnotu μ a rozptylem σ^2. Zcela ekvivalentně lze zavést náhodnou veličinu U jako přeškálovanou veličinu Y a psát

  $$U \equiv \frac{Y - m\mu}{\sqrt{m\sigma^2}} \xrightarrow{n\to\infty} N(0,1). \tag{52}$$

  Hustota pravděpodobnosti normálního rozdělení je dána vzorcem (56).

6.2 Příklady náhodných veličin

- **Rovnoměrné rozdělení $R(a,b)$ na intervalu $\langle a, b\rangle$:**

  $$\rho_R(x) = \frac{1}{b-a} = \text{konst.} \tag{53}$$
  $$\mathrm{E}\left[R\right] = \frac{a+b}{2} \tag{54}$$
  $$\sigma_R^2 = \frac{(b-a)^2}{12}. \tag{55}$$

- **Gaussovo normální rozdělení $N\left(\mu, \sigma^2\right)$:**

  $$\rho_N(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\,\mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{56}$$
  $$\mathrm{E}\left[N\right] = \mu \tag{57}$$
  $$\sigma_N^2 = \sigma^2. \tag{58}$$

- **Poissonovo rozdělení:** Diskrétní rozdělení udávající počet nezávislých jevů k v zadaném intervalu (například počet lidí, které potkáme na mostě cestou z Holešovic do Troji, nebo počet rozpadů radioaktivního prvku ve vzorku za jednotku času). Rozdělení pravděpodobnosti je

  $$P_k = \frac{\lambda^k}{k!}\,\mathrm{e}^{-\lambda}, \tag{59}$$
  $$\mathrm{E}\left[P\right] = \lambda, \tag{60}$$
  $$\sigma_P^2 = \lambda, \tag{61}$$

  přičemž parametr λ udává zároveň střední hodnotu a zároveň rozptyl rozdělení.

Úkol 6.1: *Dokažte vztahy* (54)–(55) *a* (60)–(55).

Úkol 6.2: *Naprogramujte funkci pro výpočet histogramu: na vstupu bude pole hodnot (výběr z nějakého rozdělení) a počet intervalů histogramu; na výstupu bude pole, jehož každý prvek bude odpovídat jednomu intervalu histogramu a ponese počet hodnot, které do tohoto intervalu padnou ze vstupního pole. Zamyslete se nad co nejefektivnějším algoritmem.*

Výstupní pole funkce vykreslete jako čárový graf.[9]

Úkol 6.3: *Otestujte funkci z předchozího úkolu na následujících vstupních polích:*

1. *Výběr z rovnoměrného rozdělení na intervalu $\langle 0, 1\rangle$ (v Pythonu generované pomocí* `random()` *z knihovny* `random`*, resp. pomocí* `generator.random()` *z knihovny* `numpy.random`*, jak je shrnuto v sekci 3.1).*
2. *Výběr ze* **součtu dvou** *rovnoměrných rozdělení na intervalu $\langle 0, 1\rangle$. Hustota pravděpodobnosti výsledného rozdělení je dána konvolucí (49). Vypočítejte analyticky pomocí tohoto vzorce, jak bude hustota pravděpodobnosti vypadat, a porovnejte se získaným histogramem.*
3. *Výběr ze* **součtu** *m rovnoměrných rozdělení na intervalu $\langle 0, 1\rangle$. Přesvědčte se, že již pro celkem malé m platí centrální limitní věta (51) a výsledné rozdělení se blíží normálnímu rozdělení $N(\mu, \sigma)$. Jaká bude střední hodnota μ a rozptyl σ tohoto rozdělení?*

Pro pěkné grafy volte alespoň $n = 10000$ (počet prvků výběru) a $N = 100$ (počet intervalů histogramu).

Úkol 6.4: *Na základě centrální limitní věty (52) vytvořte jednoduchý generátor čísel s normálním Gaussovským rozdělením $N(0, 1)$. Jaké je optimální hodnota m, abychom získali dostatečně přesnou aproximaci normálního rozdělení, a přitom použili co nejméně algebraických operací?*

Úkol 6.5: *Nakreslete histogram pro rozdělení hodnot v jednotlivých intervalech histogramů z úlohy 6.3. Jaké očekáváte statistické rozdělení v tomto případě?*

6.3 Výběr z neznámého rozdělení

V praxi se můžeme setkat se situací, kdy máme zadanou hustotu pravděpodobnosti či distribuční funkci nějakého komplikovaného rozdělení a chceme generovat výběr z tohoto rozdělení.

- **Známe-li hustotu pravděpodobnosti rozdělení** $\rho(x)$**:** Nejjednodušší metoda v tomto případě je vepsat funkci $\rho(x)$ do obdélníku $\langle a, b\rangle \times \langle c, d\rangle$[10], nagenerovat číslo rovnoměrně z tohoto obdélníku, a pokud padne pod křivku $\rho(x)$, vezmeme ho, v opačném případě ho zahodíme (tzv. *hit-and-miss metoda*, se kterou se potkáme v budoucnu u metody Monte-Carlo).
- **Známe-li distribuční funkci** $F(x)$**:** V tomto případě využijeme skutečnosti, že obor hodnot distribuční funkce je $F(x) \in \langle 0, 1\rangle$. Stačí tedy generovat náhodné číslo y z rovnoměrného rozdělení na intervalu $\langle 0, 1\rangle$ a číslo
$$x = F^{-1}(y), \tag{62}$$
kde F^{-1} je inverzní funkce k F, bude z rozdělení s danou F. Pokud neznáme inverzní funkci, řešíme numericky rovnici
$$F(x) = y, \tag{63}$$
která však s ohledem na vlastnosti distribuční funkce popsané v sekci 6.1 má téměř vždycky jedno a pouze jedno řešení.

[9] Knihovna `matplotlib` obsahuje funkce na přímé vykreslení sloupcového histogramu.

[10] Hustota pravděpodobnosti bývá obvykle definovaná na neomezeném intervalu. Pak je nutné určitou část funkce $\rho(x)$ oříznout. To funguje dobře v případě náhodných veličin s rychle ubývající hustotou pravděpodobnosti, jako je například Gaussovo rozdělení $N(0, 1)$, kde při oříznutí $x \in \langle -3\sigma, 3\sigma\rangle = \langle -3, 3\rangle$ zanedbáme jen 3‰ možných hodnot. V případě dlouhodosahových rozdělení, jako je například lognormální rozdělení nebo Gamma rozdělení, musíme obdélník volit velmi dlouhý, čímž se tato metoda stává velmi neefektivní.

Matematičtěji zapsáno: je-li $R = R(0,1)$ náhodná veličina s rovnoměrným rodělením na intervalu $\langle 0,1\rangle$, pak

$$X = F^{-1}(R) \tag{64}$$

je náhodná veličina s rozdělením daným distribuční funkcí F.

Úkol 6.6: *Pomocí právě popsané metody vytvořte generátor čísel s Gaussovským normálním rozdělením daným hustotou pravděpodobnosti* (56). *Porovnejte jeho rychlost s generátorem založeným na centrální limitní větě, který jste naprogramovali v úloze 6.4 a s generátorem z některé z knihoven.*[11]

Úkol 6.7: *Vytvořte generátor čísel z rozdělení daném distribuční funkcí*

$$F(x) = \frac{1}{2}\left(1 + \arctan x\right). \tag{65}$$

Jak vypadá analyticky hustota pravděpodobnosti? Nakreslete histogram a porovnejte.

Reference

[1] M.E. Muller, *A note on a method for generating points uniformly on n-dimensional spheres*, Communications of the Asociation for Computing Machinery **2**, 19 (1959).

[2] G. Marsaglia, *Choosing a Point from the Surface of a Sphere*, The Annals of Mathematical Statistics **43**, 645 (1972).

[11] Porovnání můžete provést tak, že nagenergujete větší množství čísel různými metodami, například $n = 10^6$, a změříte dobu výpočtu.