Dokumentace zápočtového programu - Aproximace řešení soustav ODR užitím Eulerovy metody

Jan Benda

Červenec 2021

1 Anotace

Program byl vytvořen za účelem přibližného řešení jistého typu soustav obyčejných diferenciálních rovnic libovolného stupně a vykreslení jejich řešení do grafu. K výpočtu používá algoritmus zvaný Explicitní Eulerova metoda a je napsán v programovacím jazyce Python. V dokumentaci jsou popsány formát vstupu a výstupu programu, jeho stavba a použité algoritmy.

2 Vstup a výstup

Pro spuštění programu spusťte soubor main.py. Veškerá komunikace mezi uživatelem a programem probíhá prostřednictvím konzole. Jelikož je vzhledem k povaze problému třeba zadat nemalé množství dat, rozdělme proces zadávání do pěti fází. Jako ukázku modelového vstupu použijeme SIR model pro šíření nemocí.

2.1 Pojmenování a hodnoty derivací

Nejprve je třeba zadat název proměnné. Dále zadáme názvy funkcí oddělené mezerami. Všechny pojmenování musí být jednoslovné výrazy, které

- nejsou klíčovým slovem v Pythonu nebo požitých modulech,
- neobsahují zakázané znaky, například opeprátory, závorky, dolar, tečka, čárka aj.,
- nezačínají na číslici.

Dále zadáme nejvyšší hodnoty derivací funkcí a to ve formátu n-tice kladných celých čísel, jejichž pořadí odpovídá pořadí názvů funkcí zadaných v předchozím kroku. Zde je ukázka možného vstupu v našem příkladu:

```
Zadejte název proměnné:
cas
Zadejte názvy funkcí:
S I R
Zadejte nejvyšši stupně derivaci:
1 1 1
```

SIR model je soustava tří rovnic prvního řádu, zadali jsem tedy tři neznáme a tři jedničky.

2.2 Přepisy rovnic

Ve druhé fázi zadáme rovnice. K jejich zápisu lze použít:

- standardní operace + - * /,
- znak ^ pro mocniny,

- kulaté závorky,
- celá i desetinná čísla,
- konstanty e, π, τ (`e, pi, tau`),
- většinu standardní (i nestandardních) matematických funkcí,
- proměnnou a derivace neznámých funkcí.

Funkce, které můžeme použít, jsou exponenciála - `exp`, logaritmus - `log`, trigonometrické funkce `sin, cos, tan`, jejich hyperboličtí dvojníci `sinh, cosh, tanh`, jejich inverze - `asin, asinh, acos,`..., odmocnina - `sqrt` a pár méně běžných funkcí - `gamma, erf`.

Chceme-li zapsat derivaci neznámé funkce, použijeme značení `nazevFunkce[n]` kde `n` je nezáporné celé číslo značící stupeň derivace[1]. Hodnotu funkce lze zapsat buďto jako nultou derivaci - `nazevFunkce[0]` nebo lze použít zápis bez závorek `nazevFunkce`.

V našem ukázkovém případě bude tato část vstupu vypadat takto:

```
Zadejte přepis
S[1] = -0.4*S[0]*I[0]/(S[0]+I[0]+R[0])
Zadejte přepis
I[1] = 0.4*S*I/(S+I+R)-0.04*I
Zadejte přepis
R[1] = 0.04*I
```

2.3 Parametry řešení

Dále je třeba specifikovat parametry řešení. Prvním je definiční obor, na kterém chceme soustavu řešit. Program v tomto případě přijímá dva různé formáty vstupu. Jsou-li na vstupu dvě čísla oddělená mezerou, budou interpretována jako začátek a konec intervalu. Druhý možný formát je zadání pouze jednoho číslo, znamenajícího konec intervalu definičního oboru. Jeho počátek bude automaticky nastaven na nulu. To je užitečné v případech, kdy se v rovnicích nevyskutuje proměnná a konečná délka intervalu pouze zajišťuje, že výpočet někdy skončí.

Nyní vložíme počáteční podmíky řešení. Pro každou neznámou funkci n-tého řádu je třeba zadat její počáteční hodnotu a počáteční hodnoty 1. až n-1. derivace. Ty jsou zadány pro každou funkci jako n-tice čísel oddělených mezerou. Po zadání dat proběhne výpočet, který však může chvíli trvat.

[1] Zde použité derivace funckí musí být menší než je nejvyšší derivace oné funkce zadaná v předchozím kroku.

Opět přikládám průvodný příklad:

```
Zadejte definiční obor řešení:
100
Zadejte počáteční stav funkce S (1. řádu):
99
Zadejte počáteční stav funkce I (1. řádu):
1
Zadejte počáteční stav funkce R (1. řádu):
0
```

2.4 Parametry vykreslení

Jakmile výpočet skončí, je třeba zadat parametry grafů a vykreslení dat. První pokyn nás vyzve k zadání parametrů grafu. Formát tohto vstupu je podobný slovníkům v Pythonu - je tvaru `parametr1: hodnota1, parametr2: hodnota2,` ... Zde je tabulka možných parametrů, jejich možných hodnot a jejich funkce.

Parametr	Typ hodnoty	Popis funkce
`nazev`	slovo	Nastaví nadpis grafu
`popisX`	slovo	Nastaví popis osy X
`popisY`	slovo	Nastaví popis osy Y
`stejneOsy`	ano/ne	Nastaví stejnou jednotku na obou osách
`polar`	ano/ne	Přepne do vykreslení v polárních souřadnicích
`xkcd`	ano/ne	Zapne vykreslení ve stylu xkcd

Není třeba zadávat všechny hodnoty, program automaticky doplní nezadané parametry. Je tedy možné odeslat prázdný vstup.

V dalším kroku zadáme různá vykreslení. Formát tohoto vstupu je následující:

```
dataX dataY slovnikParametru
```

kde `dataX, dataY` jsou názvy proměnnýné nebo derivací funkcí[2] a `slovnikParametrů` obsahuje parametry daného vykreslení.

Prvnímy dvěma argrumenty specifikujeme, kterou proměnnou nebo derivaci funkce chceme vynášet na osu X a kterou na osu Y. `slovnikParametrů` zadáme ve stejném tvaru, jako jsme zadávali slovník při nastavování parametrů grafu. Možné parametry jsou tentokrát pouze dva.

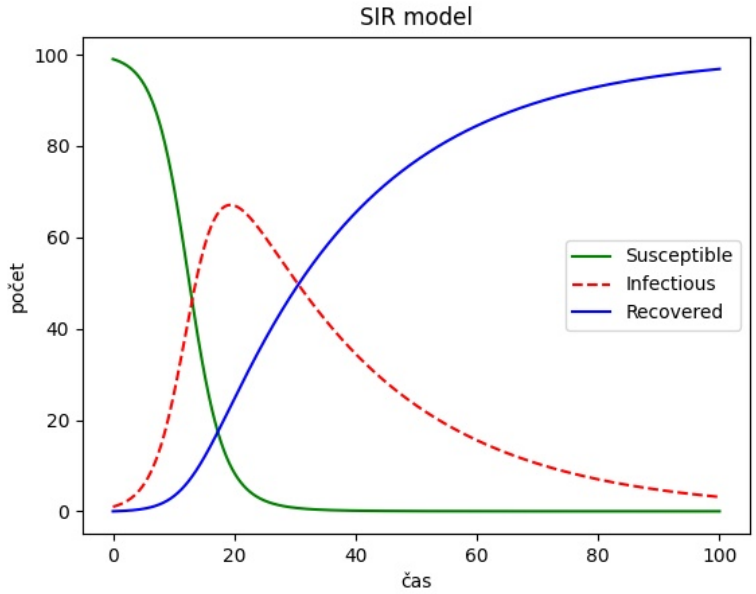
Parametr	Typ hodnoty	Popis funkce
`styl`	styl knihovny matplotlib	Nastaví barvu a styl vykreslení
`popis`	slovo	Nastaví jméno vykreslení v legendě grafu X

[2] Opět použijeme značení derivací pomocí hranatých závorek

Jakmile máme zadané všechny výkresy, odešleme prázdný řádek. Vstup v této části může vypadat nějak takto:

```
Zadejte parametry grafu:
nazev: SIR model, popisX: čas, popisY: počet
Zadejte formát a parametry vykreslení (pro vytvoření grafu odešlete prázdný vstup):
cas S styl: green, popis:Susceptible
Zadejte formát a parametry vykreslení (pro vytvoření grafu odešlete prázdný vstup):
cas I styl: r--, popis:Infectious
Zadejte formát a parametry vykreslení (pro vytvoření grafu odešlete prázdný vstup):
cas R styl: blue, popis:Recovered
Zadejte formát a parametry vykreslení (pro vytvoření grafu odešlete prázdný vstup):
```

Program v tomto momentě zanese vypočtená data do grafu, který zobrazí. V našem případě vytvoří takovýto graf.

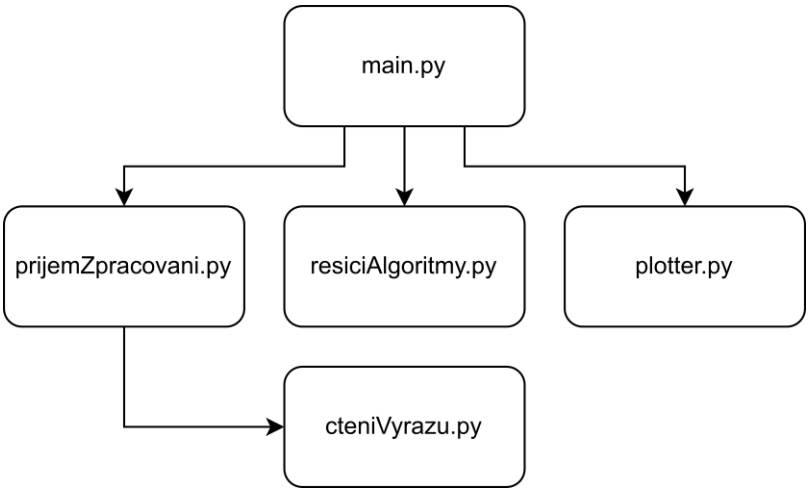


2.5 Export

Po zavření vykreslení dostaneme ještě možnost data exportovat do textového souboru. Stačí pouze kladně odpovědět a zadat jméno souboru, do kterého chceme data exportovat. Data se exportují do tabulky - první řádek tvoří popisy sloupů a zbylé řádky jsou n-ticemi čísel oddělené mezerami. Po negativní odpovědi nebo vytvoření souboru program skončí.

3 Stavba programu a použité algoritmy

Program je členěn do čtyř funkčních celků, kde každý je obsažen ve vlastním souboru, a hlavního souboru`main.py`, který je spojuje, viz diagram. O funkci každého modulu a algoritmech v něm použitých budu psát v následujících podkapitolách.



Program využívá i externí knihovny `keyword`, `math`, `numpy` a `matplotlib`, kde poslední dvě knihovny nejsou instalovány společně s Pythonem a musí být případně doinstalovány pro správné fungování programu.

3.1 prijemZpracovani.py

Soubor prijemZpracovani.py zprostředkovává komunikaci s uživatelem. Jeho funkcí je příjem vstupů, jejich kontrola a zpracování do správného formátů, vygenerování funkce pro výpočet hodnot nejvyšších derivací a export dat[3]. Kromě správného formátu vstupu jsou kontrolovány i kolize názvů funkcí a proměnných nebo zda názvy neobsahují klíčová slova Pythonu[4]

3.1.1 Generování funkce

Jediná netriviální metoda v tomto modulu je generování funkcí. Na vstupu dostane řetězce matematických funkcí zapsané v Pythoním kódu, které jsme získali předtím z modulu `cteniVyrazu.py`. Z těchto řetězců vytvoří řetězec nový, který odpovídá syntaxi definice nové funkce v Pythonu. Tento řetězec je

[3]Tato funkčnost by měla být spíše v modulu plotter.py, který se zaměřuje na výstup programu, avšak pro její přímočarost jsem ji ponechal zde.

[4]Samotné pojmenování by nevadilo, problém by vznikl při vytváření funkce, neboť je používán příkaz `exec`

zkompilován pomocí příkazu `exec`, čímž metoda vytvoří funkci `vypocetNejvyssichDerivaci`, kterou vrátí.

3.2 cteniVyrazu.py

Stringovou reprezentaci funkcí, které zadá uživatel, je třeba převést na datovou strukturu zvanou výrazový strom, což má tento modul na starosti. Tento krok je nezbytný ze dvou důvodů. Jednak umožní převést řetězcové reprezentace matematických výrazů na kód Pythonu a zároveň slouží jako přepážka mezi uživatelským vstupem a potenciálně nebezpečnou funkcí `exec`.

3.2.1 Algoritmus vytváření výrazového stromu

Algoritmus dostane na vstupu řetězec reprecentující matematický výraz. Výrazový strom je vytvářen rekurzivně podle hloubky zanoření závorek. Nejprve najdeme ty výrazy, které jsou zanořeny do závorek, zpracujeme je touto metodou a výsledný podstrom uložíme do fronty. Ve vstupním stringu je nahradíme kontrolním znakem "$".

Zbyde nám výraz bez závorek, který rozsekáme do seznamu na slova (proměnná, neznámé funkce, matematické funkce, konstanty), čísla a operátory.

V dalším kroku zpracujeme matematické funkce. Jelikož za každou funkcí následuje výraz ohraničený závorkami, bude v našem seznamu následována kontrolním znakem "$"reprezentujícím podstrom. Z funkce uděláme kořen podstromu a dáme jí jako syna výraz z dříve vytvořené fronty odpovídající pozici znaku "$". Tímto novým stromem nahradíme původní místo funkce a jejího argumentu v našem seznamu.

Nyní se postaráme o binární operátory ($+,-,*,/,\hat{}$). Pro všechny použijeme stejný postup. Zaměřme se na konkrétní operátor, např. umocnění. Budeme z našeho seznamu vytvářet nový seznam podle následujících pravidel:

- není-li položka seznamu operátor umocnění, přidej ji do nového seznamu
- je-li položka operátorem umocnění, dej ji do kořene nově vytvořeného stromu, jako levého syna jí dej poslední položku nového seznamu a jako pravého syna následující složku starého seznamu, jejíž zpracování přeskoč. Nově vytvořený strom přidej do nového seznamu.

Toto provedeme pro každý operátor podle jeho pořadí ve vyhodnocování výrazů (nejprve mocnění, poté násobení, dělení, nakonec sčítání, odčítání).

Ve finále nám zůstane v listu pouze jedna položka, kterou vrátíme jako výsledek. V případě prvního zavolání rekurze to bude výrazový strom zadaného výrazu.

3.3 resiciAlgoritmy.py

Tento soubor, jak už jeho název napovídá, má na strarosti řešení zadané soustavy. Jelikož se jedná o nejdůležitější část programu, bude jí zde v dokumentaci věnováno více místa, než zbylým částem.

3.3.1 Explicitní Eulerova metoda

Jak už bylo v anotaci zmíněno, program používá Eulerovu metodu pro aproximaci řešení soustav ODR. Ta funguje následujícím způsobem.

Uvažujme nejtriviálnější případ - jednu rovnici prvního řádu,

$$y'(x) = f(y(x), x).$$

Derivaci na levé straně rozepíšeme podle definice na

$$\lim_{\Delta x \to 0} \frac{y(x + \Delta x) - y(x)}{\Delta x} = f(y(x), x).$$

Z definice limity platí, že pokud existuje, lze její hodnotu aproximovat tak, že budeme volit hodnoty délky kroku Δx blízké nule [5]. Poté co zvolíme hodnotu Δx můžeme odstranit limitu a vyjádřit si $y(x + \Delta x)$.

$$\begin{aligned} \frac{y(x + \Delta x) - y(x)}{\Delta x} &= f(y(x), x), \\ y(x + \Delta x) &= y(x) + f(y(x), x) \cdot \Delta x. \end{aligned}$$

Iterativním použitím tohoto vzorce jsme schopni aproximovat řešení libovolné rovnice prvního stupně.

Máme-li soustavu rovnic prvního řádu, lze stejným postupem vyjádřit hodnotu každé funkce v bodě $x + \Delta x$. Výsledek bude vypadat takto.

$$\begin{aligned} y_1(x + \Delta x) &= y_1(x) + f_1(y_1(x), y_2(x), ..., y_k(x), x) \cdot \Delta x \\ y_2(x + \Delta x) &= y_2(x) + f_2(y_1(x), y_2(x), ..., y_k(x), x) \cdot \Delta x \\ &\vdots \\ y_k(x + \Delta x) &= y_k(x) + f_k(y_1(x), y_2(x), ..., y_k(x), x) \cdot \Delta x. \end{aligned}$$

Zatím jsme uvažovali pouze rovnice prvního řádu. Uvažujme rovnici n-tého řádu

$$y^{(n)}(x) = f\Big(y(x), y'(x), ..., y^{(n-1)}(x), x\Big)$$

Užitím jisté vlastnosti derivací ($y^{(n)}(x) = \big(y^{(n-1)}(x)\big)'$) lze předchozí rovnost rozepsat na soustavu

$$\begin{aligned} \Big(y^{(n-1)}(x)\Big)' &= f\Big(y(x), y'(x), ..., y^{(n-1)}(x), x\Big) \\ \Big(y^{(n-2)}(x)\Big)' &= y^{(n-1)}(x) \\ &\vdots \\ (y(x))' &= y^{(1)}(x) \end{aligned}$$

[5] Ve světě matematiky dokonce platí, že čím je Δx blíže nule, tím lepší aproximaci dostaneme, avšak později uvidíme, že ve světě programování to není vždy pravda

Pokud nebudeme brát funkce $y, y', ..., y^{(n-1)}$ jako derivace původní funkce y ale jako neznámé funkce provázané pouze zadanou soustavou rovnic, převedli jsme problém soustavy rovnic vyšších řádů na soustavu více rovnic prvního řádu, tedy předchozí případ. Analogicky bychom si poradili s víceřádovou soustavou.

3.3.2 Implementace

Dostaneme-li soustavu k rovnic, kde i-tá rovnice je n_i-tého stupně, je k popisu současného stavu třeba $1+\sum_{i=1}^{k}(n_i)$ proměnných v programu. Zavedeme si tedy pole stavů (dále pouze `stav`) v bodě x jako Pythoní seznam s následující stavbou. Nultá až $n_1 - 1$ položka reprezentují nultou až $n_1 - 1$ derivaci první funkce, n_1 až $n_1 + n_2 - 1$ položka nultou až n_2 derivaci druhé funkce, atd. Poslední položka seznamu ukládá stav proměnné. Viz znázornění níže.

$$\texttt{stav} = \left[y_1^{(0)}, ..., y_1^{(n_1-1)}, y_2^{(0)}, ..., y_2^{(n_2-1)}, ..., y_k^{(0)}, ..., y_k^{(n_k-1)}, x\right]$$

Tento modul je stejně jako celý programu psán objektově. Obsahuje třídu `Solver`, obsahující metody `krokFce`, která ze stavu v bodě x vypočítá stav v bodě $x + \Delta x$ (v této třídě nic nedělá), a `iteruj`, která předchozí metodu iteruje a průběžně ukládá hodnoty pole `stav`, obsahující informaci o řešení soustavy. Ty jsou průběžně ukládány do matice od `numpy`, kterou po dokončení výpočtu vrátí.

Dále modul obsahuje třídu `SolverEuler`, odvozenou od třídy `Solver`. Ta se liší pouze tím, že metoda `krokFce` již není nečinná, ale dopočítává stavy pomocí Eulerova algoritmu popsaného výše. Zde je použita funkce `vypocetNejvyssichDerivaci` vytvořená modulem `prijemZpracovani.py`. Jedná se o implementaci funkcí pravé strany $f_1, f_2, ..., f_k$ ze sekce o Eulerově metodě. Jako vstup bere pole `stav` a vrací pole stejného formátu v bodě $x + \Delta x$.

3.3.3 Chyba aproximace a volba délky kroku

Jak už bylo řečeno Eulerova metoda nedává přesné řešení, ale pouze aproximaci. Existují dva způsoby, jakými chyba vzniká použitím této metody.

Prvním způsobem je tzv. tronkační chyba. Lokální tronkační chyba je odchylka od exaktního řešení vzniklá v jediném kroku. Označme nově vypočtenou hodnotu jako $y_1(x) = y_0(x) + y_0'(x) \cdot \Delta x$ a proveďme polynomiální rozvoj funkce y_0 podle Taylorovy věty

$$y_0(x + \Delta x) = y_0(x) + y_0'(x) \cdot \Delta x + \mathcal{O}\big(\Delta x^2\big).$$

Rozdílem těchto dvou kvantit dostaneme odhad lokální tronkační chyby

$$E_{LT} = y_0(x + \Delta x) - y_1(x) = \mathcal{O}\big(\Delta x^2\big).$$

Celková tronkační chyba je součet všech lokálních tronkačních chyb. Budeme-li řešit soustavu na intervalu délky D, bude celkový počet kroků roven $\frac{L}{\Delta x}$. Celková chyba je tedy úměrná

$$E_{GT} = \frac{L}{\Delta x} \cdot \mathcal{O}\big(\Delta x^2\big) = \mathcal{O}(\Delta x).$$

Druhým způsobem je chyba zaokrouhlení, která vzniká neexaktním ukládáním čísel do paměti počítače. Velikost této chyby na celé délce intervalu je úměrná

$$E_R = \frac{\varepsilon}{\sqrt{\Delta x}}$$

kde ε je strojové epsilon - relativní přesnost ukládání čísel do typu float.

Lze vidět, že zatímco tronkační chyba s nižší délkou kroku klesá, chyba zaokrouhlení roste. Existuje tedy optimální délka kroku a k ní příslušná minimální možná chyba. S nižší délkou kroku zároveň roste jejich počet a tedy i délka výpočtu. Po zohlednění těchto faktorů jsem zvolil délku kroku $\Delta x = 0.001$

Pokud bychom chtěli snížit celkovou chybu aproximace při zachování délky výpočetního času, můžeme

- použít přesnější metody, např. Runge-Kuttovy metody
- použít rychlejší programovací jazyk (např. C) nebo JIT compiler

3.4 plotter.py

Poslední modul má na starosti vykreslení grafu z vypočítaných a uživtelem zadaných dat prostřednictvím knihovny `matplotlib`. Neobsahuje žádné algoritmy, které by stály za zmínění. Existuje samostatně pouze proto, aby byla oddělena vykreslovací část programu od ostatních funkčních celků.

4 Další testovací data

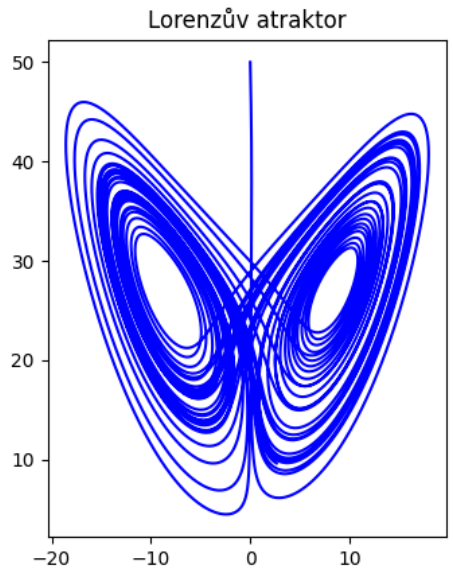
4.1 Lorenzův atraktor

Vstup:

```
Zadejte název proměnné:
t
Zadejte názvy funkcí:
x y z
Zadejte nejvyšši stupně derivaci:
1 1 1
Zadejte přepis
x[1] = 10*(y-x)
Zadejte přepis
y[1] = x*(28-z)-y
Zadejte přepis
z[1] = x*y-8/3*z
Zadejte definiční obor řešení:
40
Zadejte počáteční stav funkce x (1. řádu):
0
Zadejte počáteční stav funkce y (1. řádu):
0.3
Zadejte počáteční stav funkce z (1. řádu):
50
Probíhá výpočet. Může to chvíli trvat.
Zadejte parametry grafu:
nazev: Lorenzův atraktor, stejneOsy:ano
Zadejte formát a parametry vykreslení (pro vytvoření grafu odešlete prázdný vstup):
x z styl:blue
Zadejte formát a parametry vykreslení (pro vytvoření grafu odešlete prázdný vstup):

Chcete data exportovat do textového souboru?
ne
```

Výstup:



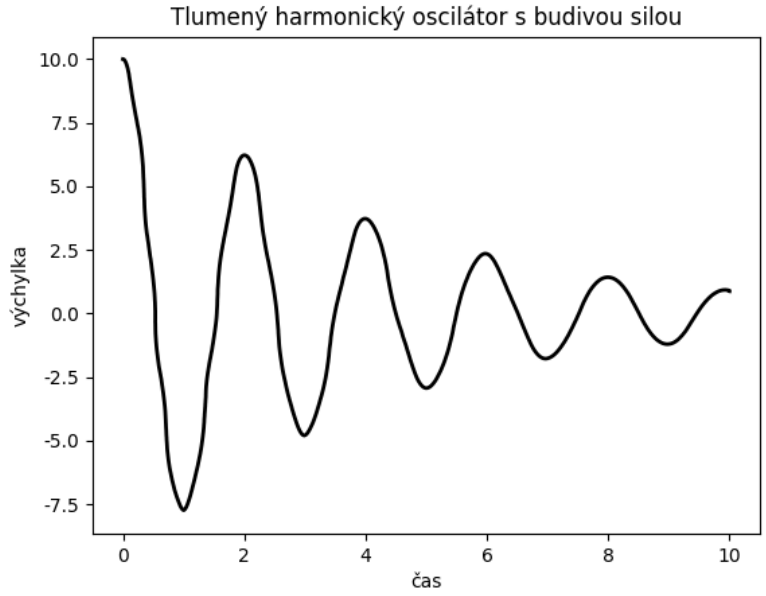
4.2 Tlumený harmonický oscilátor s budivou silou

Vstup:

```
Zadejte název proměnné:
t
Zadejte názvy funkcí:
y
Zadejte nejvyšši stupně derivaci:
2
Zadejte přepis
y[2] = -0.5*y[1]-10*y[0]+sin(5*t)
Zadejte definiční obor řešení:
10
Zadejte počáteční stav funkce y (2. řádu):
10 0
Probíhá výpočet. Může to chvíli trvat.
Zadejte parametry grafu:
nazev: Tlumený harmonický oscilátor s budivou silou, popisX: čas, popisY:výchylka, xkcd:ano
Zadejte formát a parametry vykreslení (pro vytvoření grafu odešlete prázdný vstup):
t y
Zadejte formát a parametry vykreslení (pro vytvoření grafu odešlete prázdný vstup):

Chcete data exportovat do textového souboru?
ne
```

Výstup:



Ukázka použití proměnné, derivací funkce, funkcí.

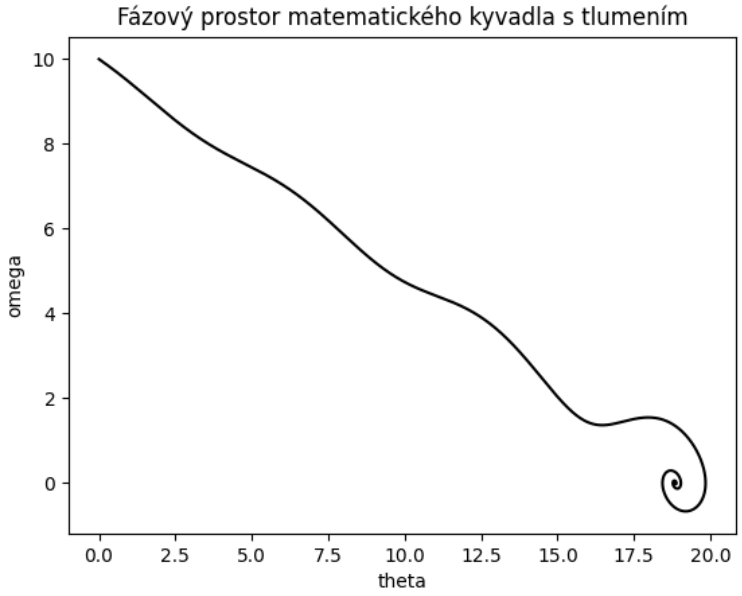
4.3 Fázový prostor matematického kyvadla s tlumením

Vstup:

```
Zadejte název proměnné:
t
Zadejte názvy funkcí:
theta
Zadejte nejvyšši stupně derivaci:
2
Zadejte přepis
theta[2] = -sin(theta)-0.5*theta[1]
Zadejte definiční obor řešení:
30
Zadejte počáteční stav funkce theta (2. řádu):
0 10
Probíhá výpočet. Může to chvíli trvat.
Zadejte parametry grafu:
nazev:Fázový prostor matematického kyvadla s tlumením, popisX:theta, popisY:omega
Zadejte formát a parametry vykreslení (pro vytvoření grafu odešlete prázdný vstup):
theta theta[1]
Zadejte formát a parametry vykreslení (pro vytvoření grafu odešlete prázdný vstup):

Chcete data exportovat do textového souboru?
ne
```

Výstup:



Zde je ukázka, proč může být užitečné vykreslovat závislosti derivace funkce na její funkční hodnotě.

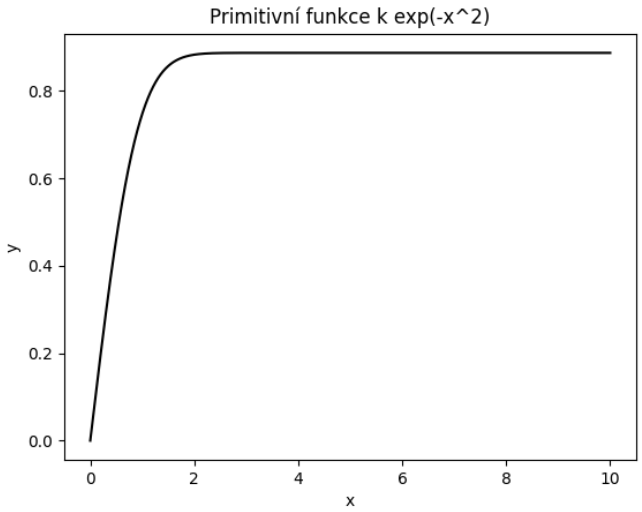
4.4 Výpočet integrálu $\int \mathbf{e}^{-x^2}\,\mathrm{d}x$

Vstup:

```
Zadejte název proměnné:
x
Zadejte názvy funkcí:
y
Zadejte nejvyšši stupně derivaci:
1
Zadejte přepis
y[1] = exp(-x^2)
Zadejte definiční obor řešení:
10
Zadejte počáteční stav funkce y (1. řádu):
0
Probíhá výpočet. Může to chvíli trvat.
Zadejte parametry grafu:
nazev:Primitivní funkce k exp(-x^2), popisX:x,popisY:y
Zadejte formát a parametry vykreslení (pro vytvoření grafu odešlete prázdný vstup):
x y
Zadejte formát a parametry vykreslení (pro vytvoření grafu odešlete prázdný vstup):

Chcete data exportovat do textového souboru?
ano
Zadejte jméno souboru:
erf.txt
```

Výstup:



Integrál $\int_0^\infty \mathbf{e}^{-x^2}\,\mathrm{d}x$ je známý tím, že neurčitý integrál je analyticky neřešitelný a určitý integrál od nuly do nekonečna je roven $\frac{\sqrt{\pi}}{2} \approx 0.8862$. Pokud otevřeme soubor a podíváme se na poslední řádek, najdeme tam přibližné hodnoty

```
10.0000 0.8867
```

Relativní chyba výpočtu byla tedy v tomto případě přibližně 0.006%.

Obsah