EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

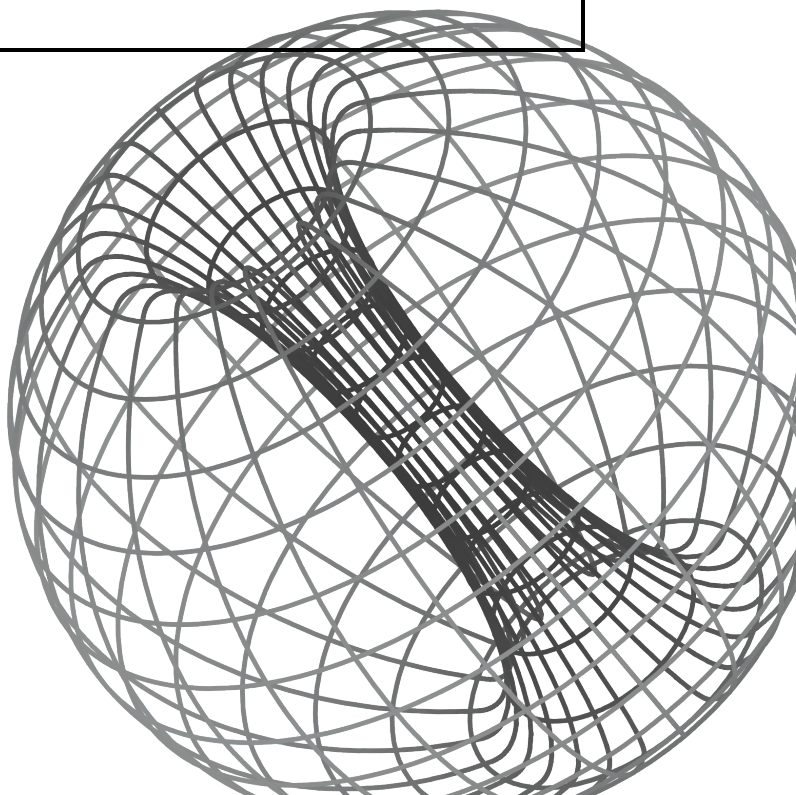
SLEZSKÁ UNIVERZITA
FYZIKÁLNÍ ÚSTAV V OPAVĚ

Zvýšení kvality vzdělávání na Slezské univerzitě v Opavě ve vazbě na potřeby Moravskoslezského kraje
CZ.02.2.69/0.0/0.0/18_058/0010238

Gabriel Török et al

Studijní opora pro předmět

Numerické metody v přírodních vědách



Fyzikální ústav
Slezská univerzita v Opavě

Opava
20. 6. 2020

Numerické metody v přírodních vědách

doc. RNDr. Gabriel Török, Ph. D

Mgr. Debora Lančová, RNDr. Jiří Horák, Ph.D, Mgr. Kateřina Klimovičová,
Mgr. Gabriela Urbancová, Mgr. Martin Urbanec, Ph.D, Mgr. Adam Hofer,
Prof. RNDr. Vladimír Karas, DrSc. a RNDr. Martin Blaschke, Ph.D

Dostupné na: physics.cz

Tato studijní opora vznikla v rámci projektu

Zvýšení kvality vzdělávání na Slezské univerzitě v Opavě
ve vazbě na potřeby Moravskoslezského kraje,

registrační číslo CZ.02.2.69/0.0/0.0/18-058/0010238

Fyzikální ústav
Slezská univerzita v Opavě
Bezručovo nám. 13, Opava

Autor šablony: RNDr. Šárka Vavrečková, Ph.D

Sázeno v systému LaTeX

Předmluva

Co najdeme v těchto skriptech

Tato studijní opora je určena studentům předmětu *Numerické modelování v přírodních vědách*. Cílem tohoto předmětu je ukázat použití různých numerických metod v praxi, zejména při řešení reálných fyzikálních problémů. Studijní opora je sestavena z 9 kapitol, kdy každá kapitola se věnuje jiné oblasti z fyziky. Předmět numerické metody v přírodních vědách má za cíl ukázat použití numerických metod v praxi, sestává tedy, stejně jako tato opora, z jednotlivých příkladů, a spíše než teoretické znalosti zde najdete použití různých metod v praxi. K studijní opoře je tedy nutné přistupovat jako ke sbírce řešených i neřešených příkladů. Ve většině kapitol naleznete také ukázku řešení části problémů v jazyce `Python`, případně `Wolfram Mathematica`.

Vzhledem ke komplexnosti zadání sestává opora z úloh, které připravil široký autorský kolektiv. Některé z nich jsou dílem studentů doktorského studia Fyzikálního ústavu, jiné vytvořili odborní asistenti zde působící. Další kapitoly byly napsány vědeckými pracovníky z Astronomického ústavu Akademie věd ČR. Proto je každá kapitola svým charakterem do jisté míry odlišná. Autoři jednotlivých kapitol budou dále nápomocní při řešení úloh, případně poskytnou další zadání pro semestrální práce.

Nedílnou součástí studijní opory jsou `Jupyter` a `Mathematica` notebooky, které Vám poskytne vyučující předmětu.

Značení

Ve skriptech se používají následující barevné ikony:

- *Rychlý náhled* (skript, kapitoly), ve kterém se dozvíme, o čem to bude.
- *Klíčová slova* kapitoly.
- *Cíle studia* pro kapitolu nám řeknou, co nového se v dané kapitole naučíme.
- Nové *pojmy*, značení apod. jsou označeny modrým symbolem, který vidíme zde vlevo. Tuto ikonu (stejně jako následující) najdeme na začátku odstavce, ve kterém je nový pojem zaváděn.
- Konkrétní *postupy* a nástroje, způsoby řešení různých situací, do kterých se může správce počítačového vybavení dostat, atd. jsou značeny také modrou ikonou.

-  Některé části textu jsou označeny fialovou ikonou, což znamená, že jde o *nepovinné úseky*, které nejsou probírány (většinou; studenti si je mohou podle zájmu vyžádat nebo sami prostudovat). Jejich účelem je dobrovolné rozšíření znalostí studentů o pokročilá témata, na která obvykle při výuce nezbývá moc času.
-  Žlutou ikonou jsou označeny odkazy, na kterých lze získat *další informace* o tématu. Nejčastěji u této ikony najdeme webové odkazy na stránky, kde se dané tématice jejich autoři věnují podrobněji.
-  Červená je ikona pro *upozornění* a poznámky.

Pokud je množství textu patřícího k určité ikoně větší, je celý blok ohraničen prostředím s ikonami na začátku i konci, například pro definování nového pojmu:

 **Definice**

V takovém prostředí definujeme pojem či vysvětlujeme sice relativně známý, ale komplexní pojem s více významy či vlastnostmi.

Podobně může vypadat prostředí pro delší postup nebo delší poznámku či více odkazů na další informace. Mohou být použita také jiná prostředí:

 **Příklad**

Takto vypadá prostředí s příkladem, obvykle nějakého postupu. Příklady jsou obvykle komentovány, aby byl jasný postup jejich řešení.

 **Úkol**

Otázky a úkoly, náměty na vyzkoušení, které se doporučuje při procvičování učiva provádět, jsou uzavřeny v tomto prostředí. Pokud je v prostředí více úkolů, jsou číslovány.

 **Souhrn kapitoly**

Na konci každé kapitoly je souhrn, ve kterém si ujasníte, o čem daná kapitola byla a co všechno jste se naučili.

Obsah

Kapitola 1

Zákon lomu a Duhová funkce

Debora Lančová, Gabriel Török

Rychlý náhled: Tato kapitola se zabývá odvozením rovnice primární a sekundární duhu, jejich vykreslením a nalezením maxima a minima. Jsou zde popsány základní metody numerické derivace a hledání kořenů funkce.

Klíčová slova: Numerická derivace, hledání kořenů funkce, metoda půlení intervalů, bisekce, duha, zákon lomu, duhová funkce, difrakce

Cíle studia: Po prostudování kapitoly budete schopní v `Pythonu` naprogramovat jednoduchou numerickou derivaci funkce jedné proměnné, použít a implementovat metodu půlení intervalů (bisekce) – společně budete schopni najít extrémy funkcí. Seznámíte se také s tím, jak se vykreslují jednoduché grafy pomocí knihovny `matplotlib`.

Duha je optický jev, který vzniká odrazem slunečního světla na kapkách deště. Vzniká jen za specifických podmínek, kapky musí mít správnou velikost, Slunce musí být nízko nad obzorem, a jiné.

1.1 Jak vzniká duha - fyzikální základ

Abychom mohli nastínit základní princip vzniku duhy, musíme nejprve zavést některá zjednodušení - například, že dešťové kapky jsou kulové. To víceméně platí pro malé kapky, o velikosti $0.5 - 0.7$ mm, jejichž tvar je výrazně ovlivněn povrchovým napětím. Předpokládejme tedy, že Sluneční paprsky dopadají na kulatou kapku vody a při dopadu se lámou. Bude přitom používat pouze zákony geometrické optiky.

1.1.1 Lom světla

Zákon lomu, jinak také Snellův zákon, říká, jakým způsobem se mění chod optického paprsku při přechodu mezi optickými prostředími. Optické vlastnosti prostředí jsou vyjádřeny *indexem lomu* n, který říká, o kolik pomaleji se v něm šíří světlo oproti rychlosti šíření ve vakuu

$$n = \frac{c}{v}, \tag{1.1}$$

kde v je rychlost šíření světla v daném prostředí. Indexy lomu v různých prostředích jsou shrnuty v tabulce 1.1.

Obrázek 1.1: Dvojitá duha viditelná z budovy Fyzikálního ústavu v Opavě, včetně výrazného Alexandrova tmavého pásu

Tabulka 1.1: Indexy lomu v různém prostředí.

Prostředí	Vzduch	Voda	Sklo	Diamant
Index lomu	1.00	1.33	1.5	2.42

Podle *Fermatova principu nejkratšího času* se světlo šíří po takové dráze, která odpovídá nejkratšímu času. Přechází-li tedy z jednoho prostředí do druhého, ve kterých se liší rychlost šíření, bude se řídit tímto principem. Zákon lomu má tvar

$$n_1 \sin\alpha = n_2 \sin\beta, \tag{1.2}$$

kde nesmíme zapomenout, že úhly α a β se měří od kolmice (jak je nakresleno na obrázku 1.2).

1.1.2 Úplný odraz, mezní úhel

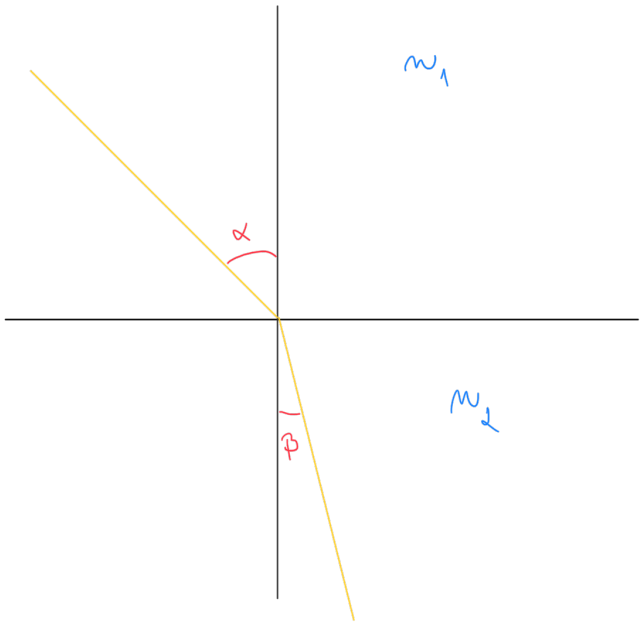
Mezní úhel dopadu je takový úhel, při němž se světlo láme pod úhlem 90° od kolmice. Pře jeho překročení se tedy všechny dopadající paprsky odrážejí - dochází k *úplnému odrazu*.

Odvození mezního úhlu, který značíme zpravidla α_m je jednoduché. Chceme-li, aby $\beta = 90°$, platí $\sin\beta = 1$ a pak

$$\sin\alpha_m = n_2/n_1, \tag{1.3}$$

Úkol

Spočítejte mezní úhly pro úplný odraz paprsku, který přechází ze vzduchu do vody a naopak.



Obrázek 1.2: Snellův zákon – zákon lomu

Tabulka 1.2: Indexy lomu světla ve vodě v závislosti na vlnové délce.

Vlnová délka	Barva	Index lomu ve vodě	Index lomu ve skle
396,85	tmavě fialová	1.343	1,471
430,78	fialová	1.341	1,468
486,14	modrá	1.337	1,464
527,00	zelená	1.335	1,461
589,30	žlutá	1.333	1,459
656,28	oranžová	1.331	1,457
686,72	červená	1.330	1,456
760,82	tmavě červená	1.329	1,454

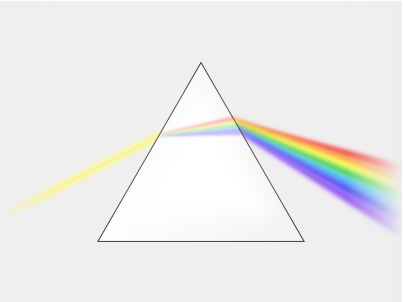
1.1.3 Barvy duhy, disperze

Disperze znamená rozklad bílého světla na jeho jednotlivé barevné složky. K tomu dochází při lomu nebo odrazu díky tomu, že pro každou vlnovou délku je hodnota indexu lomu jiná - viz tabulka 1.2. Platí, že pro rostoucí frekvenci klesá rychlost šíření v daném prostředí, a tím pádem také index lomu. Nejznámějším příkladem disperze je lom na optickém hranolu, kde dochází k rozkladu světla hned dvakrát - obrázek 1.3.

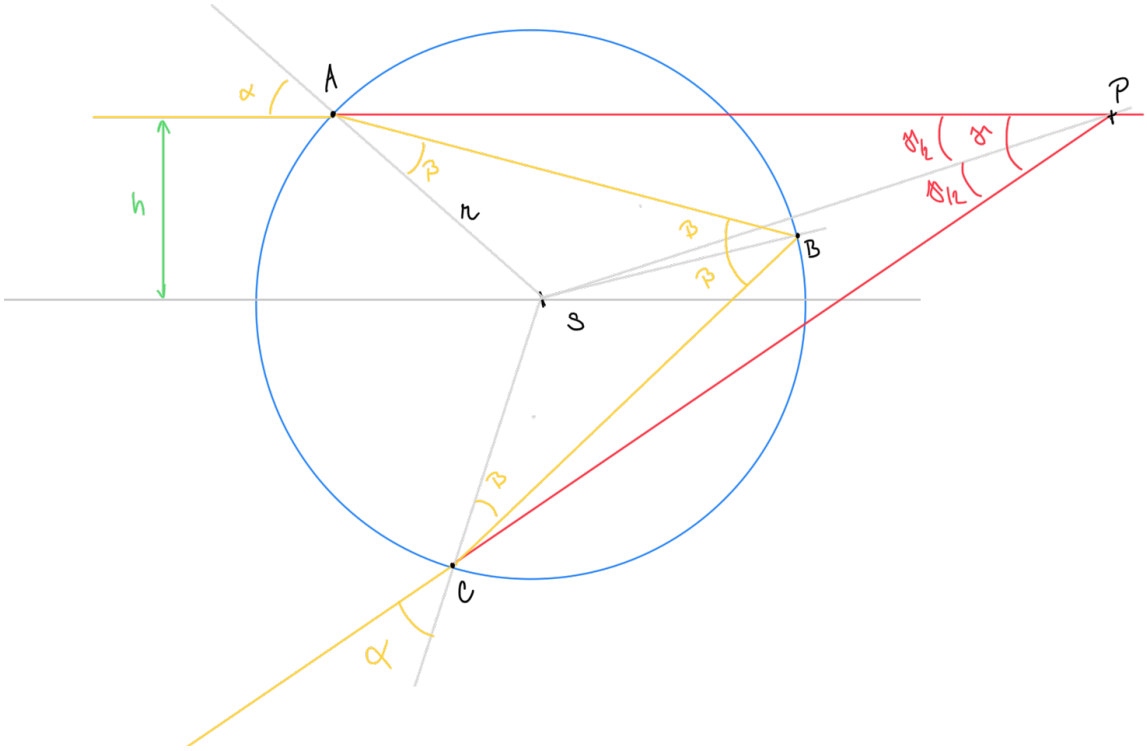
1.1.4 Průchod paprsku vodní kapkou

Nyní si tedy můžeme představit, jak vypadá situace, kdy na kulovou vodní kapku dopadne paprsek světla (obrázek 1.4). Mějme vodní kapku o poloměru r, do které v bodě A vstupuje paprsek ve vzdálenosti h od osy procházející středem S (veličině h také říkáme *impaktní parametr*). Paprsek dopadá pod úhlem α od kolmice (přímky spojující body S a A), láme se na povrchu kapky pod úhlem β. Poté se odrazí od protější stěny kapky v bodě B, dopadá opět na povrch v bodě C, láme se a opouští kapku.

Zajímá nás, pod jakým úhlem se paprsek od kapky odrazí - tedy úhel γ. Když se pečlivě podíváte na



Obrázek 1.3: Rozklad světla na optickém hranolu [Wikimedia]



Obrázek 1.4: Průchod paprsku vodní kapkou a vznik duhy

obrázek, uvidíte, že platí

$$\gamma = 4\beta - 2\alpha. \tag{1.4}$$

Úkol

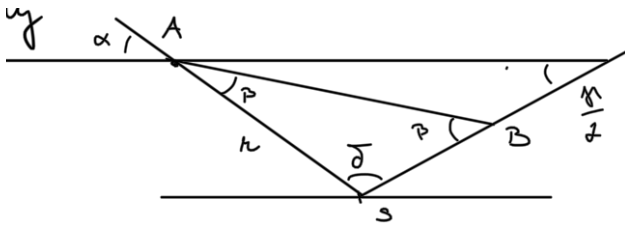
Odvoďte rovnici (1.4) z obrázku 1.4 tak, že si z ΔABS spočítáte velikost úhlu δ a pak z ΔAPS úhel γ.

Úhly α a β můžeme zapsat následovně

$$\sin\alpha = \frac{h}{r} \tag{1.5}$$

$$\sin\beta = \frac{1}{n}\sin\alpha = \frac{1}{n}\frac{h}{r}, \tag{1.6}$$

kde vztah pro úhel β vyplývá ze zákonu lomu.



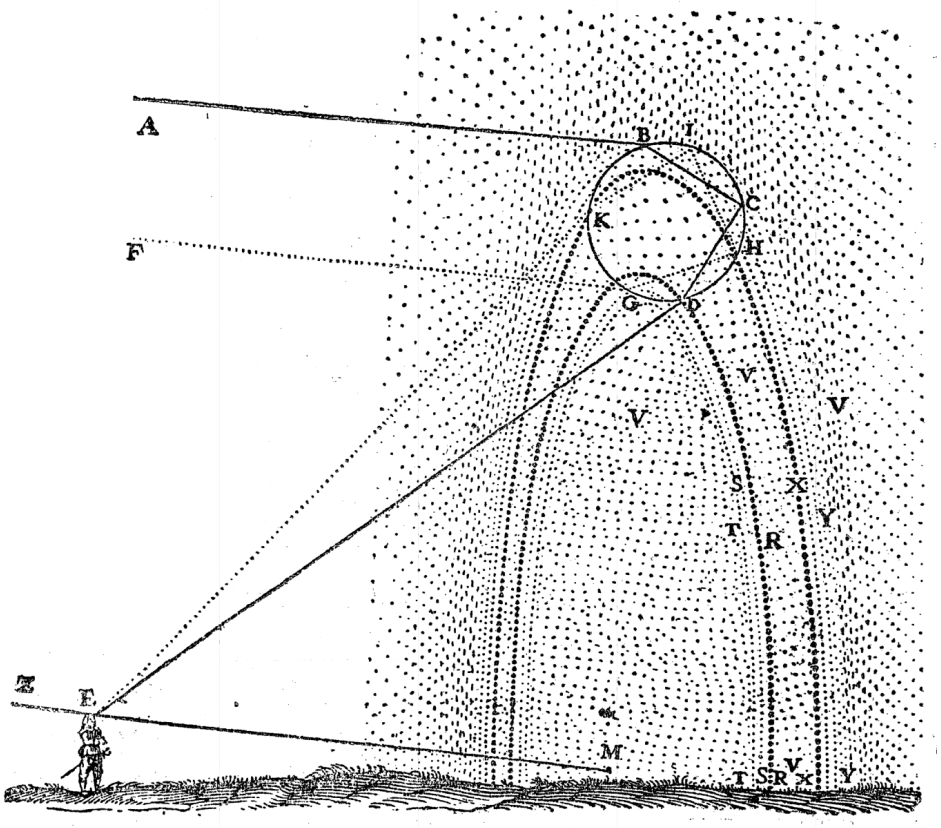
Obrázek 1.5: Průchod paprsku vodní kapkou, detailní označení úhlů a bodů.

Definice

Po dosazení rovnic (1.5) a (1.6) do rovnice (1.4) získáme finální vztah

$$\gamma = 4\arcsin\frac{h}{nr} - 2\arcsin\frac{h}{r}, \tag{1.7}$$

který označujeme jako *Duhová funkce* 1. řádu, která popisuje závislost úhlu, pod kterým se paprsek od vodní kapky odrazí.



Obrázek 1.6: Princip vzniku duhy, jak je popsal ve své knize *Discours de la méthode* z roku 1637 René Descartes

1.2 Sekundární duha a Alexandrův tmavý pás

Často můžeme pozorovat duhy hned dvě, přičemž druhá duha bude mít převrácené barvy a bude mnohem méně výrazná (jako na obrázku 1.1). Sekundární duha vniká po dvojitém odrazu paprsku uvnitř kapky, přičemž paprsek do ní vstupuje pod osou a vystupuje nad osou (obrázek 1.6).

Definice

Duhová funkce pro sekundární duhu má tvar

$$\gamma_2 = \pi - 6\arcsin\frac{h}{nr} + 2\arcsin\frac{h}{r}. \tag{1.8}$$

Úkol

Zkuste si vzorec pro sekundární duhu odvodit sami, stejným postupem jako v případě primární duhy.

Mezi primární a sekundární duhou se nachází tzv. *Alexandrův tmavý pás*, který je mnohem tmavší než zbytek oblohy. Proč tomu tak je, uvidíte, až si vykreslíme primární a sekundární duhovou funkci.

1.3 Implementace

Naším úkolem bude implementovat funkce pro primární a sekundární duhu v `Pythonu`, vykreslit si je pro různé vlnové délky a nalezení maxima a minima pomocí derivace.

1.3.1 Vykreslení duhové funkce 1. a 2. řádu

Nejprve si pouze vykreslíme obě duhové funkce, abychom viděli jejich průběhy a mohli si tak lépe představit vznik duhy.

Začněme importováním knihoven `Numpy` pro numerické výpočty a `Matplotlib` pro tvorbu grafů:

```
1 import numpy as np
2 from matplotlib import pyplot as plt
```

Nyní můžee definovat a vykreslit funkci pro jednu vlnovou délku, tedy jeden index lomu. Definujme také průměr kapky, typicky $d = 0.5\,\mathrm{mm}$.

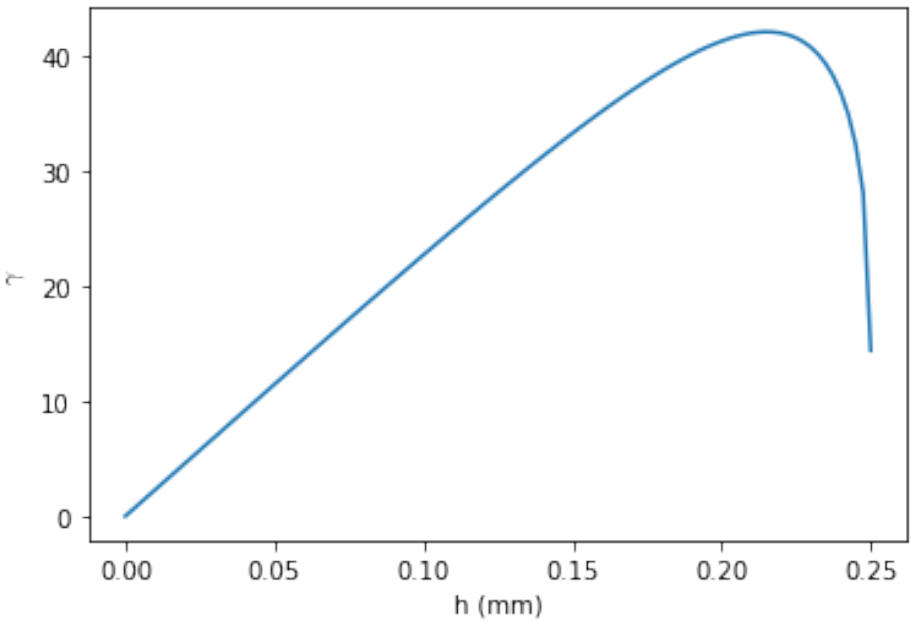
```
1 #index lomu světla
2 n = 1.3333
3 #typicky prumer destove kapky je 0.5 - 0.7 mm
4 r = 2.5 * 1e-4
5 #duhova funkce
6 def gama(x,r):
7     foo = 4.0 * np.arcsin(1/n * x/r)
8     bar = 2.0 * np.arcsin(x/r)
9     return foo - bar
```

Chceme vykreslit graf závislosti γ na impaktním parametru h, musíme tedy vytvořit `array` s hodnotami h od 0 (horizontální osa kapky) do $h = r$, využijeme funci `linspace` z knihovny `numpy`. Pak do další `array` `gamy` uložíme hodnoty funkce `gama(h,r)` pro všechny prvky `h`:

```
# impaktni parametry od nuly do r, celkem 100 hodnot
h = np.linspace (0,r, num = 100)
gamy = gama(h,r)
```

Nyní už můžeme primární duhovou funkci vykreslit. Avšak, nesmíme zapomenout na převody jednotek. h jsme si zaveli v metrech, knihovna `numpy` počítá gonieometrické funkce v radiánech - převedeme si je do stupňů:

```
plt.plot(h*1000,gamy*180/np.pi)
plt.xlabel('h (mm)')
plt.ylabel(r'$\gamma$')
plt.show()
```



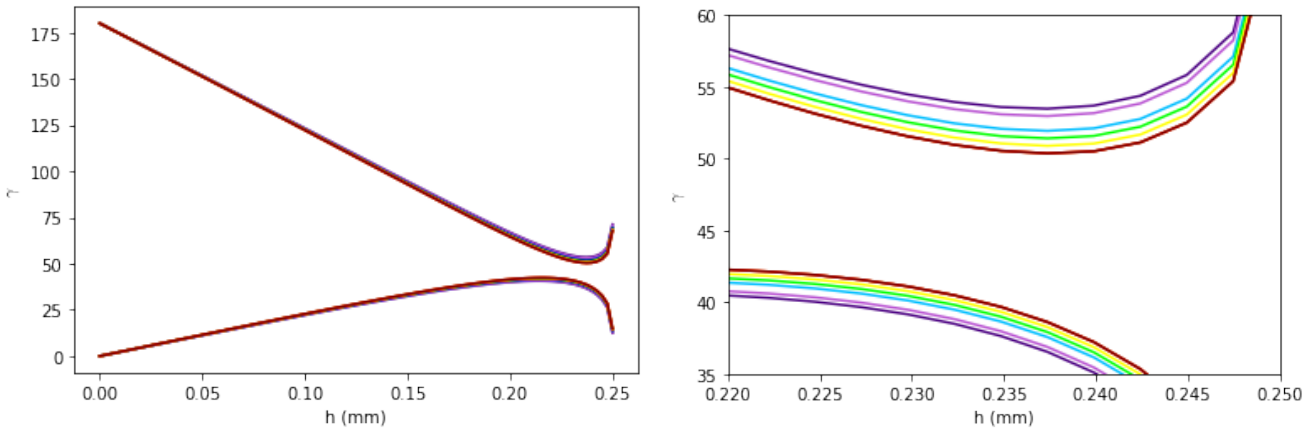
Obrázek 1.7: Duhová funkce primární duhy pro index lomu $n = 1.33$

Výsledek můžete vidět na obrázku 1.7. Nyní už jen podle pohledu můžete vidět, že nejvíce paprsků se od kapky odrazí pod úhlem přibližně 40° - tam právě dochází ke vzniku primární duhy. Jaká je konkrétní hodnota tohoto maxima najdeme pomocí numerické derivace v další kapitole. Nyní pojďme do grafu přidat barvy a funkci pro sekundární duhu (všimněte si, že jsme převod z radiánů do stupňů přidali již přímo do funkce pro γ).

```
def gama_n(x,r,n):
    foo = 4.0 * np.arcsin(1/n * x/r)
    bar = 2. * np.arcsin(x/r)
    return (foo - bar) * 180/np.pi

```

```
6  def gama_2(x,r,n):
7      foo = 6.0 * np.arcsin(x/(n*r))
8      bar = 2.0 * np.arcsin(x/r)
9      return (np.pi - foo + bar) * 180/np.pi
10
11 indexy = [1.343,1.341,1.337,1.335,1.333,1.331,1.330,1.329]
12 barvy = ['indigo', 'mediumorchid', 'blue', 'lime', 'yellow', 'orange', 'red', 'maroon']
13
14 for i in range(0,len(indexy)):
15     plt.plot(h*1000,gama_n(h,r,indexy[i]),color = barvy[i])
16     plt.plot(h*1000,gama_2(h,r,indexy[i]),color = barvy[i])
17
18 plt.xlabel('h (mm)')
19 plt.ylabel(r'$\gamma$')
20
21 plt.show()
```

Všimněte si, že jsme pro vykreslení grafu použili smyčky `for`. Také jsme zvolili barvy tak, aby víceméně odpovídali barvě duhy – knihovna `Matplotlib` nabízí celou řadu barev, přehledně je naleznete na jejich webových stránkách[1]. Výsledný graf je na obrázku 1.8, včetně výřezu nejzajímavější části kolem maxima primární a minima sekundární funkce. Nyní už pravděpodobně chápete, jak vzniká Alexandrův tmavý pás – mezi primární a sekundární duhou je oblast, do které se odráží žádné paprsky – podle zákonů geometrické optiky by tedy měl být úplně černý. Vidíte také, proč má sekundární duha otočené barvy oproti primární duze.



Obrázek 1.8: Duhová funkce pro primární a sekundární duhu – celé (vlevo) a přiblížení na oblast maxima a minima (vpravo).

Úkol

Jak se změní vlastnosti duhové funkce 1. a 2. řádu pro jinou velikost kapky?

[1] `https://matplotlib.org/3.1.0/gallery/color/named_colors.html`

1.3.2 Numerické derivace, hledání minima a maxima funkce

I když můžeme použít vestavěné funkce a knihovny, pojďme si funkci, která bude počítat derivaci, implementovat sami.

Definice

Derivace funkce $f(x)$ podle x je limita

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} = \lim_{\delta x \to 0} \frac{f(x+\delta x) - f(x)}{\delta x} \tag{1.9}$$

Derivaci můžeme použít k nalezení extrémů funkce. Jelikož derivace funkce je rovna směrnici tečny funkce v daném bodě, pro extrém funkce bude platit, že derivace v tomto místě je rovná nule. Toho využijeme při hledání úhlu, pod kterým se od kapky odrazí nejvíce paprsků. Při pouhém pohledu na graf na obrázku 1.8 vidíme, že tato hodnota se bude pohybovat okolo 40°, ale pojďme najít přesnou hodnotu.

Použijeme nejjednodušší metodu, zvolíme $h \ll 1$ a řekneme, že derivace funkce $f(x)$ v bodě x_0 je

$$d(x_0) = \frac{f(x_0+h) - f(x_0)}{h}. \tag{1.10}$$

```
1 def derivative(f,x,h=0.01):
2         return (f(x + h) - f(x))/h
```

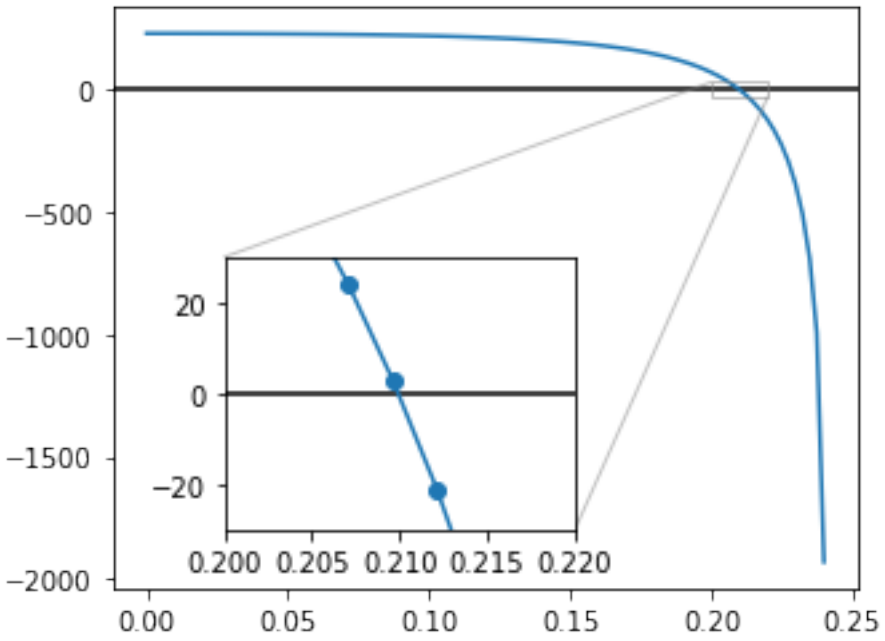
Spočítáme derivace funkce (1.7) podle h a podíváme se na její průběh. Budeme hledat maximum třeba pro žlutou barvu ($n = 1.333$). Nadefinujeme si ale duhovou funkci znova, pomocí `lambda` funkce. Ta je užitečná, pokud máme velmi krátké funkce, může obsahovat pouze 1 výraz. Jejich výhoda spočívá v anonymnosti – nemají název, hodí se, pokud chcete rychle vytvořit jednoduchou funkci, krátce jí používat a poté zase zapomenout.

```
1 n = 1.333
2 r = 0.25
3 hs = np.linspace(0,r,100)
4 gammafce = lambda x: (4.0 * np.arcsin(1/n * x/r) - (2. * np.arcsin(x/r))) * 180/np.pi
5 gama_der = derivative(gammafce,hs)
6 gamas = gammafce(hs)
7 plt.axhline(0,color ='k')
8 plt.plot(hs,gama_der)
9 plt.show()
```

Budeme hledat takovou hodnotu γ, kde platí $\frac{\mathrm{d}\gamma}{\mathrm{d}h} = 0$, tedy průnik derivace s osou y. Na obrázku 1.9 je znázorněn průběh derivace duhové funkce a zvýrazněna je osa y. V přiblížení vidíme, že funkce protne nulu přibližně kolem hodnoty $h = 0,21$, avšak získat přesné číslo je komplikovanější – musíme použít další numerickou metodu, *půlení intervalů (bisekce)*.

1.3.3 Hledání kořene - metoda půlení intervalů

Nejjednodušší metoda pro nalezení kořene rovnice se nazývá metoda půlení intervalů. Spočívá v jednoduchém předpokladu – ve chvíli, kdy funkce překročí osu y, změní se znaménko její funkční hodnoty. Její princip je velmi jednoduchý:



Obrázek 1.9: Průběh derivace duhové funkce. V obrázku je také přiblížení, které v kódu nenajdete - je jen ilustrační.

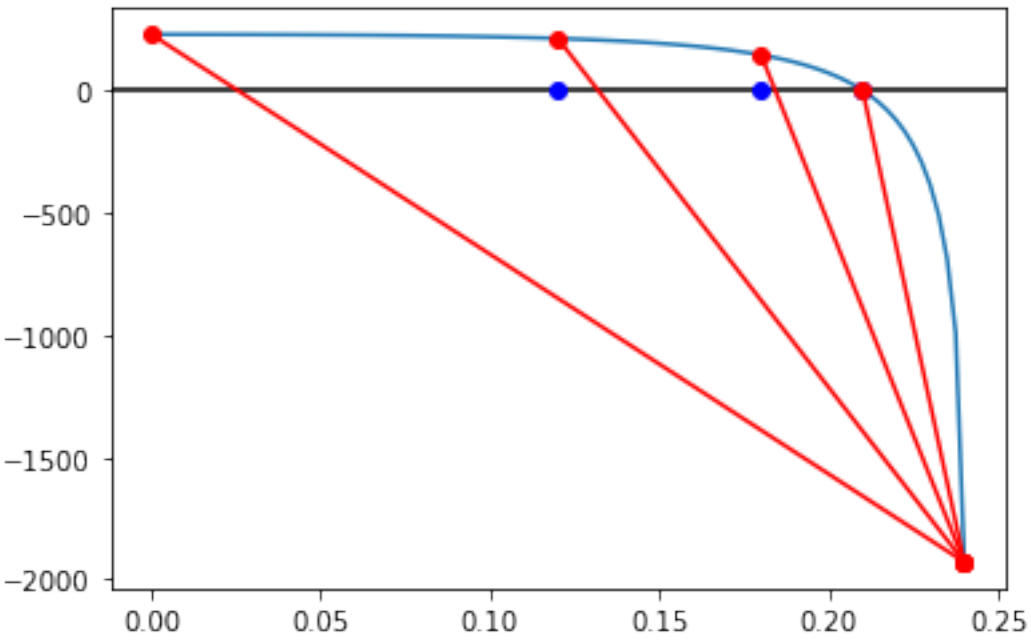
Postup

- Vezmeme 2 funkční hodnoty $f(x_0)$ a $f(x_1)$. Pokud platí, že $\operatorname{sgn}(f(x_0)) = -\operatorname{sgn}(f(x_1))$, víme, že se kořen nachází někde v intervalu mezi x_1 a x_0. Pokud to neplatí, musíme interval rozšířit.
- Určíme funkční hodnotu v bodě x_2, který leží přesně mezi body x_1 a x_0, $x_2 = \frac{x_0+x_1}{2}$.
- Zjistíme, zda má funkční hodnota $f(x_2)$ stejné znaménko, jako $f(x_0)$ nebo jako $f(x_1)$. Podle toho určíme nové dva okrajové body tak, aby měly funkční hodnoty opačné znaménko.
- Pokračujeme stejným způsobem, dokud nenajdeme takové x_n, kde platí $f(x_n) = 0$ (nebo alespoň $f(x_n) = 0 \pm \epsilon$, kde $\epsilon \ll 1$).

Vyzkoušíme si to tedy na naší derivaci duhové funkce. Využijeme opět lambda funkci a vytvoříme funkci `der`, která počítá přímo derivaci duhové funkce

```
#Puleni intervalu
der = lambda x: derivative(gammafce,x)
x0 = hs[0]
x1 = hs[-5]
xs = np.array([x0,x1])
epsilon = 0.001
vysledek = False
plt.axhline(0,color ='k')
plt.plot(hs,der(hs))
while (vysledek == False):
```

```
    x2 = (x0 + x1)/2.
    xs = np.array([x0,x1])
    plt.plot(xs,der(xs),'-or')
    if((der(x2) < np.abs(0.0+epsilon))):
        vysledek = x2
        break
    plt.plot(x2,0,'-ob')
    if (np.sign(der(x0)*der(x2)) == 1): x0 = x2
    elif (np.sign(der(x1)*der(x2)) == 1): x1 = x2
plt.show()
print(vysledek, gammafce(vysledek))
```



Obrázek 1.10: Hledání kořene pomocí metody půlení intervalů.

Výsledek, který nám kód vypočítá, je hodnota $\gamma = 41.45°$.

Úkoly

1. Spočítejte, jaké jsou maxima pro ostatní barvy podle tabulky 1.2.
2. Spočítejte minimum duhové funkce sekundární duhy pro různé barvy. Určete úhlovou šířku Alexandrova tmavého pásu.
3. Existuje také duha třetího a čtvrtého řádu, kterou ale pozorujeme na opačné straně, tedy kolem Slunce. Jsou už opravdu velmi vzácné. Zkuste odvodit, jakým způsobem se v takovém případě paprsky v kapkách odrážejí a jakou mají duhovou funkci.

Souhrn kapitoly

V této kapitole jsme nastínili základní principy numerického řešení fyzikálních problémů a základní vykreslování grafů pomocí knihovny `matplotlib` v Pythonu. Naučili jsme se jednoduché metody numerické derivace a hledání kořenů, které budou dále rozvedeny také v následujících kapitolách.

Otázky

1. Proč je v Alexandrův pás tmavý?
2. Jaké jsou nevýhody metody bisekce?
3. Proč jsou barvy sekundární duhy otočené oproti primární?
4. Jaké pořadí barev bude v duze 3. a 4. řádu?

Další informace

- Encyklopedie fyziky, Duha, J. Reichl
- Optické úkazy v atmosféře na www.ukazy.astro.cz
- Duha 3. a 4. řádu (anglicky)

Kapitola 2

Modelování dat

Kateřina Klimovičová, Gabriel Török

Rychlý náhled: V kapitole si povíme něco o modelování dat. Především se zaměříme na metodu nejmenších čtverců.

Klíčová slova: Modelování dat, metoda nejmenších čtverců.

Cíle studia: Po prostudování kapitoly by jste měli být schopni používat metodu nejmenších čtverců.

Mějme dáno N bodů

$$[x_1; y_1], [x_2; y_2], ...[x_N; y_N], \tag{2.1}$$

kde $x_1, x_2, ...x_N$ jsou pevně dané hodnoty veličiny x. $y_1, y_2, ...y_N$ jsou hodnoty získané měřením veličiny y. Někdy se setkáváme s označením nezávisle (veličina x) a závisle (veličina y) proměnná.

Příklad

Veličinou y může být poloha vozidla jedoucího konstantní rychlostí po rovné silnici (označme x'). Veličina x může být čas měření daného x' (označme t). Máme N bodů

$$[t_1; x'_1], [t_2; x'_2], ..., [t_N; x'_N]. \tag{2.2}$$

Hodnota x'_1 je poloha vozidla měřená v čase t_1 atd.

Předpokládejme, že y je známou funkcí x. Neboli, že platí

$$y = f(x). \tag{2.3}$$

Příklad

V našem příkladu máme

$$x' = f(t) = v_0 t + x'_0, \tag{2.4}$$

kde v_0 a x'_0 jsou konstanty (rychlost a poloha vozidla v čase $t = 0$).

Pokud jsme schopni veličinu y měřit přesně, platí

$$y_1 = f(x_1), y_2 = f(x_2), ..., y_N = f(x_N). \tag{2.5}$$

Jinými slovy, všechny body v rovině $x - y$ leží na křivce dané vztahem $y = f(x)$.

Běžně se však setkáváme se situací, kdy veličinu y nejsme schopni měřit přesně.

Funkce $f(x)$ obecně závisí na P parametrech. Předpokládejme, že počet parametrů P je menší než počet bodů N. Označme parametry jako $a_1, a_2, ..., a_P$. Můžeme napsat $y(x) = f(x; a_1, a_2, ..., a_P)$. Platí

$$y_i = f(x_i; a_1, a_2, ..., a_P) + e_i. \tag{2.6}$$

Veličiny $e_1, e_2, ..e_N$ jsou náhodné (chyby měření).

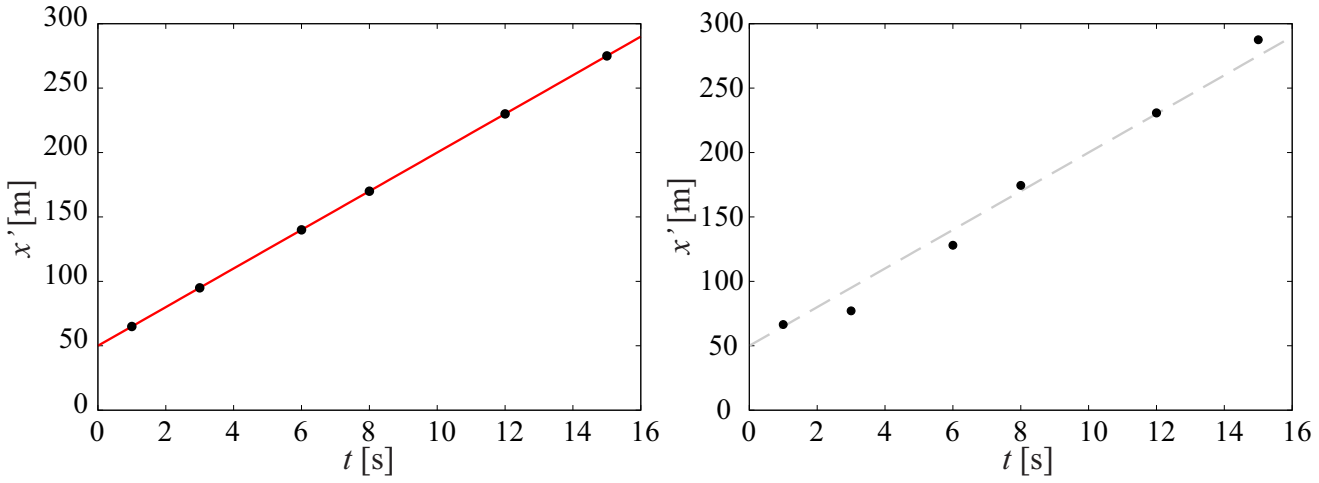
Příklad

Vraťme se k našemu příkladu. Pokud polohu vozidla měříme přesně, graf závislosti polohy x' na čase t může vypadat jako ten umístěný na levém panelu obr. 2.1. Pokud polohu nejsme schopni naměřit zcela přesně, graf bude vypadat jako ten na pravém panelu stejného obrázku. Pro jednotlivé x'_i platí

$$x'_i = f(t_i; v_0, x'_0) + e_i, \tag{2.7}$$

kde

$$f(t_i; v_0, x'_0) = t_i v_0 + x'_0. \tag{2.8}$$



Obrázek 2.1: Poloha vozidla x' jako funkce času měření t. Levý panel: Poloha vozidla měřená přesně. Černé body odpovídají jednotlivým měřením. Červená křivka je dána rovnicí (2.7). Parametry jsme zvolili: $v_0 = 15\,\mathrm{m.s^{-1}}$ a $x'_0 = 50\,\mathrm{m}$. Pravý panel: obdoba levého panelu pro případ kdy rychlost není měřena přesně.

Pokud bychom měli veličinu y změřenou přesně, pak proložit ji křivkou $y = f(x; a_1, a_2, ...a_P)$ není problém. Pokud nejsme schopni veličinu y měřit přesně pak musíme nejdříve odhadnout hodnoty zatím neznámých parametrů $a_1, a_2, ..., a_P$. Přitom požadujeme aby křivka co nejlépe přiléhala k bodům $[x_1; y_1], [x_2; y_2], ..., [x_N; y_N]$. Jelikož nalezené hodnoty nebudou přesnými hodnotami parametrů, ale pouze jejich odhady, je třeba si zavést nové označení. Například je můžeme značit velkým písmenem $(A_1, A_2, ...A_P)$. Dále je třeba zvolit kritérium určující jak moc křivka přiléhá k bodům.

2.1 Metoda nejmenších čtverců

V případě metody nejmenších čtverců je kritériem přiléhavosti křivky k bodům minimálnost veličiny

$$S = \sum_{i=1}^{N} [y_i - f(x_i; A_1, A_2, ...A_p)]^2. \tag{2.9}$$

Jedná se o součet kvadrátů (čtverců) rozdílu mezi y_i a $f(x_i; A_1, A_2, ...A_p)$.

Odhady $A_1, A_2, ..., A_p$ získáme z podmínky

$$\sum_{i=1}^{N} [y_i - f(x_i; A_1, A_2, ...A_p)]^2 = \min\ (S) \tag{2.10}$$

Příklad

V našem příkladu jedoucího vozidla je

$$x' = v_0 t + x'_0 \tag{2.11}$$

Neboli

$$S = \sum_{i=1}^{N} [x'_i - v_0 t - x'_0]^2. \tag{2.12}$$

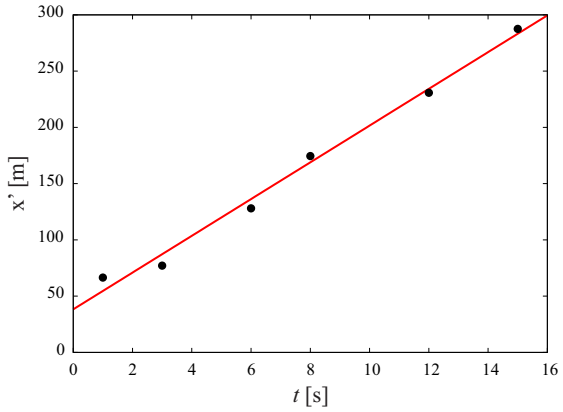
Odhady na V_0 a X'_0 získáme z podmínky

$$\sum_{i=1}^{N} [x'_i - V_0 t - X'_0]^2 = \min\ (S). \tag{2.13}$$

Nyní aplikujeme výše uvedený postup na body (viz pravý panel obr. 2.1):

$$[1; 66, 52], [3; 77, 14], [6; 128, 01], [8; 174, 53], [12; 230, 79], [15; 287, 52]. \tag{2.14}$$

Jinými slovy najdeme takové V_0 a X'_0, pro které veličina S nabývá minima. Tímto postupem dostaneme $V_0 = 16,3297\,\mathrm{m.s^{-1}}$ a $X'_0 = 38,27890\,\mathrm{m}$. K výpočtu jsme použili program jehož přepis je uveden v kapitole 2.2. Odpovídající křivku je možné vidět na obrázku 2.2.



Obrázek 2.2: Poloha vozidla x' jako funkce času měření t proložená křivkou nejlépe přiléhající k bodům. Křivka je dána rovnicí (2.7) s parametry $V_0 = 16,3297\,\mathrm{m.s^{-1}}$ a $X'_0 = 38,27890\,\mathrm{m}$.

2.1.1 Lineární regresní funkce

Nyní se zabývejme případem, kdy $y = f(x; a_1, a_2, ...a_P)$ je tvaru

$$y = a_1\phi_1(x) + a_2\phi_2(x) + ... + a_P\phi_P(x). \tag{2.15}$$

$\phi_1(x), \phi_2(x), ...\phi_P(x)$ jsou funkce x, avšak neobsahují žádné další parametry. Jinými slovy, y „závisí“ na parametrech $a_1, a_2, ..., a_P$ lineárně.

Pro y_i obecně platí

$$y_i = a_1\phi_1(x_i) + a_2\phi_2(x_i) + ... + a_p\phi_p(x_i) + e_i, \tag{2.16}$$

kde e_i jsou náhodné veličiny. Odhad parametrů $A_1, A_2, ..., A_P$ určíme z podmínky

$$S = \sum_{i=1}^{N}[y_i - a_1\phi_1(x_i) - a_2\phi_2(x_i) - ... - a_p\phi_p(x_i)]^2 = \min S. \tag{2.17}$$

K výpočtu můžeme použít rovnic

$$\frac{\partial S}{\partial A_1} = 0, \quad \frac{\partial S}{\partial A_2} = 0, \quad ... \quad , \quad \frac{\partial S}{\partial A_P} = 0.$$

Neboli

$$\frac{\partial S}{\partial A_1} = (-2)\sum_{i=1}^{N}[y_i - A_1\phi_1(x_i) - A_2\phi_2(x_i) - ... - A_P\phi_P(x_i)]\phi_1(x_i) = 0,$$
$$\frac{\partial S}{\partial A_2} = (-2)\sum_{i=1}^{N}[y_i - A_1\phi_1(x_i) - A_2\phi_2(x_i) - ... - A_P\phi_P(x_i)]\phi_2(x_i) = 0,$$
$$\vdots$$
$$\frac{\partial S}{\partial A_P} = (-2)\sum_{i=1}^{N}[y_i - A_1\phi_1(x_i) - A_2\phi_2(x_i) - ... - A_P\phi_P(x_i)]\phi_P(x_i) = 0.$$

Zavedeme substituce

$$B_{jh} = \sum_{i=1}^{N}\phi_h(x_i)\phi_j(x_i), \quad B_h = \sum_{i=1}^{N} y_i\phi_h(x_i).$$

Dostáváme soustavu p lineárních rovnic o p neznámých:

$$B_{11}A_1 + B_{21}A_2 + ... + B_{P1}A_P = 0,$$
$$B_{12}A_1 + B_{22}A_2 + ... + B_{P2}A_P = 0,$$
$$\vdots$$
$$B_{1P}A_1 + B_{2P}A_2 + ... + B_{PP}A_P = 0.$$

Řešením jsou odhady parametrů $A_1, A_2, ..., A_P$.

Příklad

V příkladu, který nás provází celou kapitolou, je funkce

$$x' = v_0 t + x'_0 \tag{2.18}$$

skutečně lineární ve výše uvedeném smyslu.

Již výše bylo uvedeno, že

$$S = \sum_{i=1}^{6} \left(x_i - v_0 t_i - x'_0\right)^2 . \tag{2.19}$$

Dostáváme

$$\frac{\partial S}{\partial v_0} = (-2)\left(\sum_{i=1}^{6} x_i t_i - v_0 \sum_{i=1}^{6} t_i t_i - x'_0 \sum_{i=1}^{6} t_i\right) = 0, \tag{2.20}$$

$$\frac{\partial S}{\partial x'_0} = (-2)\left(\sum_{i=1}^{6} x_i - v_0 \sum_{i=1}^{6} t_i - x'_0 \sum_{i=1}^{6}\right) = 0. \tag{2.21}$$

$$\tag{2.22}$$

Po úpravě máme dvojici rovnic

$$\sum_{i=1}^{6} x'_i t_i - v_0 \sum_{i=1}^{6} t_i t_i - x'_0 \sum_{i=1}^{6} t_i = 0, \tag{2.23}$$

$$\sum_{i=1}^{6} x'_i - v_0 \sum_{i=1}^{6} t_i - x'_0 \sum_{i=1}^{6} = 0. \tag{2.24}$$

Zaveďme substituce

$$B_1 = \sum_{i=1}^{6} t_i x'_i, \quad B_2 = \sum_{i=1}^{6} x'_i, \quad B_{11} = \sum_{i=1}^{6} t_i t_i, \quad B_{12} = \sum_{i=1}^{6} t_i, \quad B_{22} = \sum_{i=1}^{6} = 6. \tag{2.25}$$

Rovnice se zredukují do tvaru

$$B_1 - v_0 B_{11} - x'_0 B_{12} = 0, \tag{2.26}$$

$$B_2 - v_0 B_{12} - x'_0 B_{22} = 0. \tag{2.27}$$

Řešením rovnic dostáváme

$$x'_0 = 38.27890\,\mathrm{m}, \quad v_0 = 16.3297\mathrm{m.s}^{-1}. \tag{2.28}$$

K výpočtu byl použit program uvedený v kapitole 2.2.

2.1.2 Příklad z reálného světa - Určení stáří vesmíru

V roce 1929 Edwin Hubble uvedl (viz [3]), že čím je galaxie vzdálenější od Země, tím se od ní rychleji vzdaluje[1]. Zjištění lze shrnout v podobně tzv. Hubbleova (či Hubbleova-Lemaîterova) zákona

$$v = Hr. \tag{2.29}$$

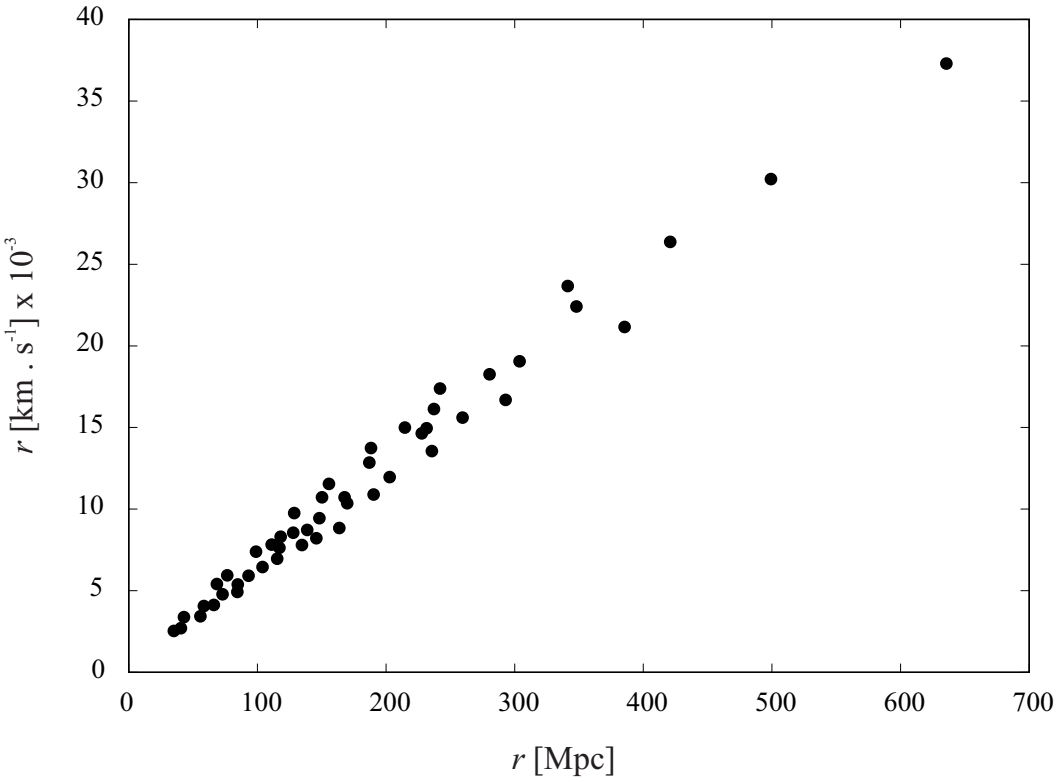
[1] Ponechme nyní stranou spory o skutečném prvenství této práce.

Hubbleův zákon dává do souvislosti rychlost vzdalování galaxie v a vzdálenost galaxie r od země. Konstanta úměrnosti, H, se nazývá Hubbleova konstanta. Může být interpretována jako relativní rychlost rozpínání vesmíru.

Z hodnoty Hubbleovy konstanty je teoreticky možné určit stáří vesmíru. Potřebujeme však vědět nejen současnou hodnotu H, ale i vývoj s časem. Předpokládejme nyní, že hodnota H se s časem nemění. Jinými slovy, že vesmír se už od svého počátku rozpíná stejně rychle.

Úkol

Na konci této podkapitoly jsou uvedeny rychlosti a vzdálenosti galaxií od Země. Vaším úkolem je určit z těchto pozorování hodnotu Hubbleovy konstanty a z ní předpokládané stáří vesmíru. Pokud si s tím nebudete vědět rady níže uvádíme podrobnější návod. Závislost rychlosti vzdalování na vzdálenosti od Země je vykreslena na obr. 2.3.



Obrázek 2.3: Hubbleův diagram. Závislost rychlosti vzdalování galaxií v na jejich vzdálenosti od Země r. Hondoty jsme převzali z [5].

Hodnoty rychlostí v galaxií meřených v km.s^{-1} a vzdálenosti galaxií r v megaparsecích (založeno na [5]):

$$\begin{aligned} r_i &= [35.101, 40.501, 42.864, 55.689, 58.39, 66.152, 68.515, 72.903, 76.615, \\ &\quad 84.378, 84.716, 93.153, 98.891, 103.954, 111.041, 115.429, 117.117, \\ &\quad 118.129, 127.917, 128.592, 134.667, 138.717, 145.805, 148.168, 150.193, \\ &\quad 155.593, 163.693, 167.743, 169.769, 186.982, 188.332, 190.357, 202.845, \\ &\quad 214.658, 227.821, 231.533, 235.583, 237.271, 241.996, 259.547, 280.473, \\ &\quad 292.96, 303.761, 341.225, 347.975, 385.439, 420.878, 499.18, 635.535] \\ v_i &= [2535.6, 2704.8, 3380.5, 3429.7, 4057.1, 4130, 5408.8, 4781.9, 5940.1, \\ &\quad 4927.4, 5385.8, 5917.2, 7389.2, 6448.7, 7824.3, 6956.1, 7631.7, 8307.3, \\ &\quad 8549.2, 9755.5, 7801.8, 8718.9, 8212.7, 9443.3, 10722.1, 11542.7, \\ &\quad 8841.2, 10723.3, 10361.5, 12847.6, 13740.4, 10893.7, 11956.1, 14996.7, \\ &\quad 14635.7, 14949.6, 13550.6, 16132.2, 17387, 15602.9, 18258.2, 16690.9, \\ &\quad 19055.9, 23666.5, 22412.4, 21160.5, 26374, 30215.3, 37293.5] \end{aligned}$$

Poznámka

Rada: Předpokládejte, že rychlost vzdalování známe přesně (i když tomu tak není) a vzdálenost je měřena s chybou. Jak spočítat stáří vesmíru? Při velkém třesku by vzdálenost mezi všemi objekty ve vesmíru měla jít do nuly. Tedy můžeme říci, že pro vzdálenost galaxií v čase t platí

$$r = vt + r_0, \qquad \text{kde } r_0 = 0. \tag{2.30}$$

Z výše uvedeného je patrné, že čas, za který se galaxie dostanou do současného stavu, vypočteme z rovnice

$$t = \frac{r}{v} = \frac{1}{H}. \tag{2.31}$$

Předpokládané stáří vesmíru je tedy nepřímo úměrné hodnotě Hubbleovy konstanty. Dávejte si však pozor na jednotky.

Řešení: Mě vyšlo stáří vesmíru přibližně 15.7 mld let .

2.2 Příklady v Pythonu

Níže uvádíme několik příkladů, jak lze řešit výše uvedené úkoly.

1. Hledání parametrů pro které křivka nejlépe přiléhá k bodům. Viz sekce 1.1.

```
1  from scipy.optimize import minimize
2
3  def square(a,b, data):
4     x, y = data
5     return ((y-(a+b*x))*(y-(a+b*x))).sum()
6
```

```
d = numpy.array([1, 3, 6, 8, 12, 15])
v = numpy.array([66.5248, 77.1354, 128.014, 174.528, 230.793,  287.515])

data = (d,v)

def squarekon(p):
    x,y=p
    return square(x,y,data)

minimize(squarejedna,(10.0,10.0))
```

2. Obdobně jako 1., avšak tentokrát je výpočet proveden metodou popsanou v sekci 1.2.

```
import numpy as np
from scipy.optimize import minimize

d = numpy.array([1, 3, 6, 8, 12, 15])
v = numpy.array([66.5248, 77.1354, 128.014, 174.528, 230.793,  287.515])

data = (d,v)

A = np.array([[-(d*d).sum(), -(d).sum()], [-(d).sum(), -6]])
B = np.array([-(d*v).sum(), -(v).sum()])
X = np.linalg.solve(A,B)

print(X)
```

3. Příklad kódu o kterém jsme dosud nehovořili, ale který může sloužit ke kontrole výše uvedených výsledků. Zde jsme použili knihovnu Kapteyn

```
import numpy
from kapteyn import kmpfit

def residuals(p, data):
   x, y = data
   a, b = p
   return (y-(a+b*x))

d = numpy.array([1, 3, 6, 8, 12, 15])
v = numpy.array([66.5248, 77.1354, 128.014, 174.528, 230.793,  287.515])

paramsinitial = [1.0, 1.0]
fitobj = kmpfit.Fitter(residuals=residuals, data=(d,v))
fitobj.fit(params0=paramsinitial)
print('Parametry:        ', fitobj.params)
```

Další informace

- Kapteyn Astronomical Institute. Least squares fitting with kmpfit. Dostupné z https://www.astro.rug.nl/software/kapteyn/kmpfittutorial.html
- PRESS William H., TEUKOLSKY Saul A., VETTERLING William T., FLANNERY Brian P. Numerical recipes. The art of scientific computing. Third edition. *Cambridge*. 2007, ISBN-13 978-0-521-88068-8
- REKTORYS, Karel a kolektiv. Přehled užité matematicy. *SNTL*. 1980, ISBN 04-003-81.

Souhrn kapitoly

V kapitole jsme se věnovali především využití metody nejmenších čtverců.

Otázky

1. Vysvětlete důvody, proč jsme na počátku předpokládali, že počet bodů N je větší než počet parametrů P.

Kapitola 3

Interpolace a extrapolace

Gabriela Urbancová

Rychlý náhled: V této kapitole se naučíme, co je interpolace a extrapolace, jak je využívat v praxi, tedy jak spojit zadanou množinu bodů křivkou tak, aby tato křivka byla hladká, procházela všemi zadanými body a co nejvíce se blížila původní, námi hledané, funkci. Uvedeme si zde příklady různých interpolací, k jejichž výpočtu a k tvorbě grafů využijeme programovací jazyk `Python` a jeho vestavěné knihovny a funkce. V těchto krátkých programech se naučíte porovnávat různé interpolační metody, jak numericky, tak graficky. Taky si ukážeme i jiný způsob, jak ověřit kvalitu zvolené interpolace, konkrétně pomocí Riemannova určitého integrálu.

Klíčová slova: Interpolace, extrapolace, interpolační uzel, interpolace polynomem, stupeň polynomu, koeficient polynomu, lineární interpolace, Lagrangeova metoda, lineární splajn, kubický splajn, `Python`, `Scipy`, `quad`, Riemannův určitý integrál, Rungeova funkce.

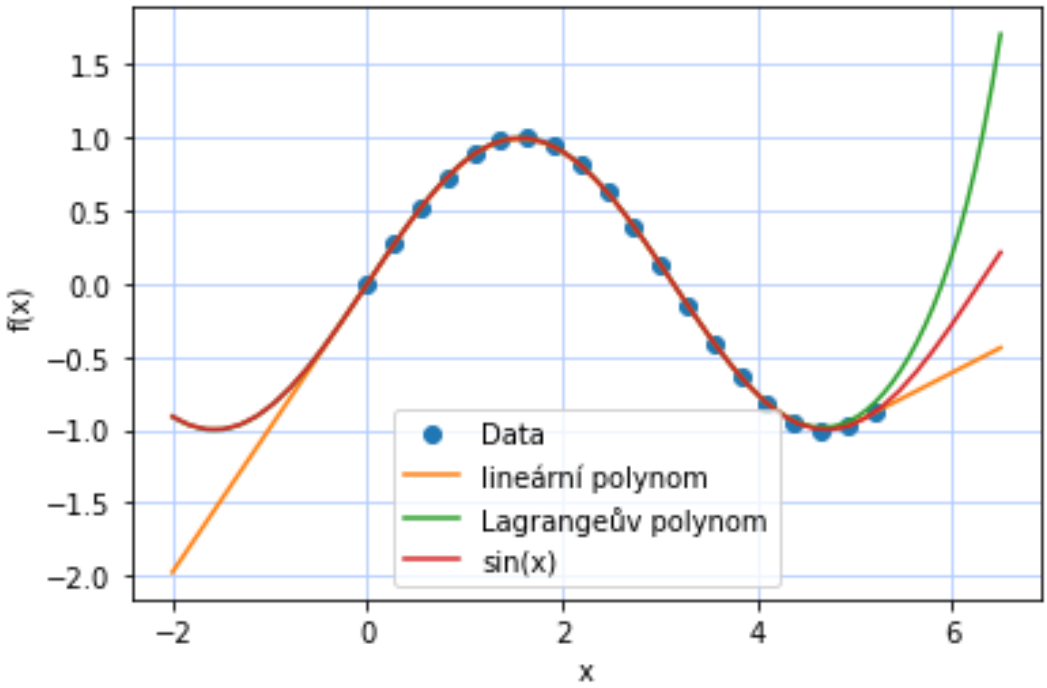
Cíle studia: Po prostudování této kapitoly budete vědět, k čemu a jak se používá interpolace a extrapolace. Budete znát několik druhů interpolací, v jakých případech se dají použít a kdy je lepší zvolit jinou aproximační metodu. Také budete umět zvolit vhodné knihovny a funkce v programu `Python`, spočítat a vykreslit jednotlivé interpolace a srovnat jejich výsledky. Nakonec si spočítáte integrály známých funkcí a srovnáte přesné výsledky s jejich interpolovanými hodnotami.

3.1 Co je to interpolace a extrapolace

Jednou z aproximačních metod, která se hojně využívá v numerické matematice, je interpolace, resp. extrapolace. Hlavním znakem interpolace je, že nově sestrojená aproximační funkce $g(x)$ musí přesně procházet všemi známými body. Tato množina bodů, tzv. interpolačních uzlů, je výsledkem původní funkce $f(x)$. Ve většině případů je to tak, že tato původní funkce $f(x)$ je neznámá a není spočítána ve všech bodech. Mějme tedy jednoduchou tabulku obsahující napočítané hodnoty y_i náležící jednorozměrné funkci $f(x_i)$ v bodech

$$x_0, x_1, ..., x_N, \ x_i \neq x_j \text{ pro } i \neq j, \tag{3.1}$$

ale my potřebujeme zjistit hodnotu y v jiném bodě x, a to tak, aby $x_0 < x < x_N$. Interpolací se tedy snažíme nalézt hodnoty, které se nachází mezi dvěma interpolačními uzly uvnitř známého intervalu.



Obrázek 3.1: Extrapolace funkce $f(x) = \sin(x)$ na 20ti interpolačních uzlech.

Někdy se stane, že nám některé okrajové body v požadovaném intervalu chybí a potřebujeme je znát. Takovému procesu se říká extrapolace a snažíme se nalézt přibližné hodnoty funkce mimo interval známých hodnot. Funguje na stejném principu jako interpolace, tedy opět hledáme vhodnou náhradní funkci. Díky této funkci, která odpovídá okrajovým hodnotám nám neznámé funkce, můžeme určit i hodnoty mimo původní rozsah měření, ale pouze s jistou pravděpodobností. Čím dál od okrajových bodů extrapolujeme, tím více nepřesné máme výsledky (roste odchylka od skutečných hodnot). Extrapolace je metoda méně využívaná, protože není tak přesná a funguje spíše jako hrubý odhad toho, jak by se mohla funkce chovat, když je za okrajovými uzly. Ukázku extrapolované funkce $f(x) = \sin(x)$ můžete vidět na obr. 3.1, použili jsme lineárního a Lagrangeův interpolační polynom. Na námi uvedeném příkladě se odhad minulého ani budoucího chování pomocí extrapolace moc nezdařil, i když jsem zvolili poměrně hodně interpolačních uzlů.

Definice

Interpolace je tedy takový typ aproximace, u které požadujeme, aby hledaná funkce přesně procházela zadanými body neboli hledáme takovou funkci $g(x)$, která se shoduje s funkcí $f(x)$ v uzlových bodech x_i

$$g(x_i) = f(x_i) = y_i \quad i = 1, \ldots, N. \tag{3.2}$$

Existuje celá řada různých typů interpolací [6], u některých požadujeme i shodnost derivací v uzlových bodech, tj. $g'(x_i) = f'(x_i)$, u jiných i spojitost druhých derivací (např. Hermiteova interpolace). Každá z existujících interpolací má své výhody i omezení.

V tomto textu si uvedeme některé základní interpolace [7], omezíme se na případ jedno dimenzionální, kdy všechny body leží v rovině v kartézském souřadném systému a jsou seřazeny vzestupně podle první souřadnice. Samozřejmě se dají interpolovat i funkce více proměnných, ale pro zachování jednoduchosti

této kapitoly se jimi zabývat nebudeme.

Na konci této kapitoly najdete části kódu psaného v `Pythonu`, pomocí kterého si můžete vyzkoušet druhy interpolací, které zde uvedeme.

3.2 Polynomiální interpolace

Při interpolování se snažíme najít takovou třídu funkcí, která bude v nějakém smyslu výhodná. Nejvíce využívané funkce v běžné praxi jsou ty ve tvaru polynomu

$$P(x) = A_0 + A_1 x + A_2 x^2 + \ldots + A_N x^N = \sum_{i=1}^{N} A_i \cdot x^{N-i}. \tag{3.3}$$

Požadujeme, aby byla splněna interpolační podmínka (3.2)

$$P_N(x_i) = y_i \quad i = 1, \ldots, N \tag{3.4}$$

a dostáváme tedy předpis pro interpolační polynom nejvýše stupně N.

Jak můžeme vidět, tak matematický tvar polynomu vypadá jednoduše. Stačí, když určíme stupeň polynomu, napočítáme příslušné koeficienty pomocí řešení soustavy lineárních rovnic a dostaneme jednoznačný tvar hledaného interpolačního polynomu. Stupeň polynomu je definován jako nejvyšší mocnitel proměnné v nenulovém členu. Výhodou tohoto tvaru funkce je také následné počítání derivace a integrace. Volbou různého stupně polynomu jsme schopni dosáhnout libovolné přesnosti, ale pozor, u vysokých stupňů polynomu už dochází k velkých zaokrouhlovacím chybám a k nežádoucím oscilacím.

3.2.1 Lineární interpolace

Polynomem prvního stupně (pro $N = 1$) interpolujeme lineárně. Dva sousední uzly se jednoduše proloží úsečkou. Hodnotu interpolační funkce v bodě x spočítáme tak, že nejdříve zjistíme, mezi kterými uzly námi hledaný bod leží. Je potřeba vědět, zda mám uzly rozmístěny rovnoměrně či nerovnoměrně. Na tento výpočet existují různé algoritmy. Když nakonec znám polohu x_i a x_{i+1}, námi hledaná interpolační funkce se spočítá ze vztahu

$$\frac{g(x) - f(x_i)}{x - x_i} = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \quad \Rightarrow \quad g(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}(x - x_i). \tag{3.5}$$

Takto vzniklá interpolační funkce je sice spojitá, ale není hladká, tedy její první derivace není spojitá, proto je její použití značně omezené. Na obr. 3.2 vidíme srovnání lineární a polynomiální interpolace..

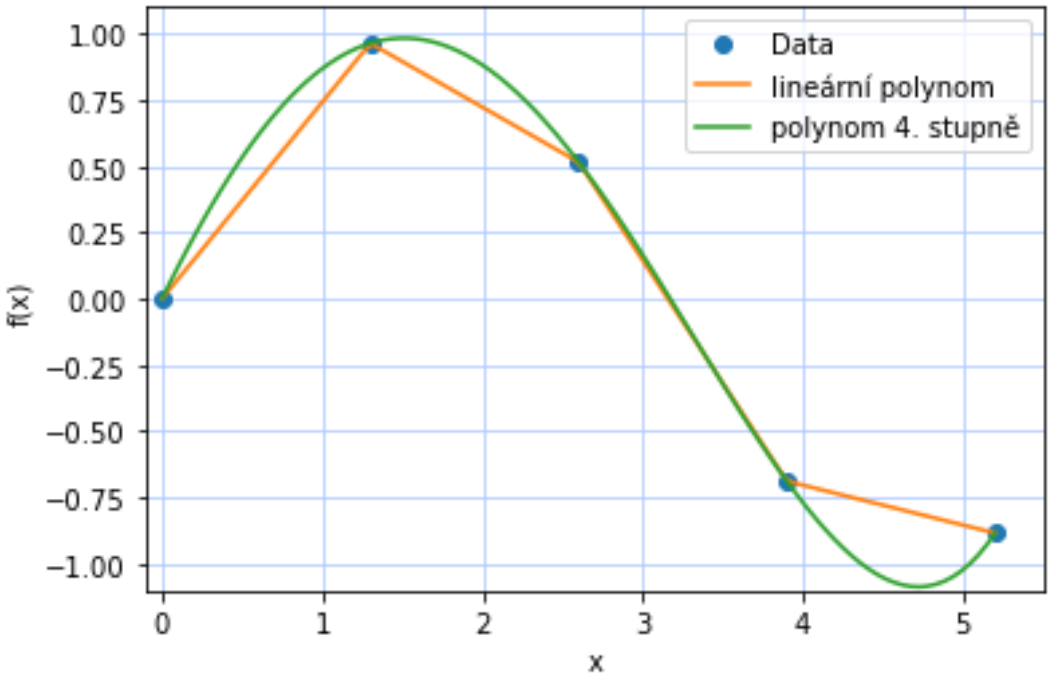
3.2.2 Lagrangeův interpolační polynom

Při výpočtu jednotlivých koeficientů polynomu může docházet ke vzniku velkých zaokrouhlovacích chyb, hlavně v případech vysokého stupně polynomu. Lepším řešením je se tomuto výpočtu úplně vyhnout a použít tzv. Lagrangeovu metodu. Ta spočívá ve složení jednotlivých elementárních polynomů

$$l_i(x) = \frac{(x - x_0)\ldots(x - x_{i-1})(x - x_{i+1})\ldots(x - x_N)}{(x_i - x_0)\ldots(x_i - x_{i-1})(x_i - x_{i+1})\ldots(x_i - x_N)} \tag{3.6}$$

splňujících podmínku

$$l_i(x_j) = \delta_i j, \tag{3.7}$$



Obrázek 3.2: Srovnání lineární a polynomiální interpolace pro případ 4 interpolačních uzlů.

do výsledného Lagrangeova interpolačního polynomu

$$L_N(x_k) = \sum_{i=0}^{N} y_i l_i(x_k) = y_k, \quad \text{pro } i, k = 0, \ldots, N. \tag{3.8}$$

Pokud máme tabulku obsahující vysoký počet interpolačních uzlů, kterými výsledný polynom musí projít, tak Lagrangeova metoda není vhodná, její výpočet se stává složitý a zdlouhavý. Naopak je metoda dobře využitelná, pokud interpolujeme různé funkce, ale ve stejných uzlových bodech.

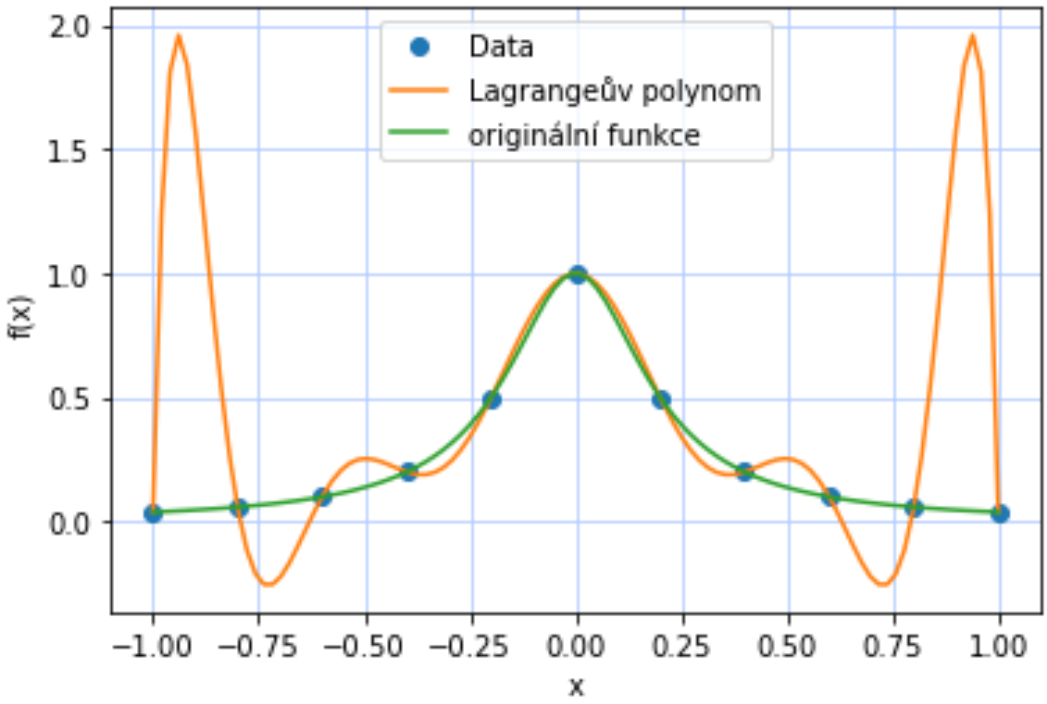
3.3 Interpolace pomocí splajnu

Jak už jsme uvedli dříve, v případech vysokého počtu interpolačních uzlů musíme k interpolaci zvolit polynom vysokého stupně. Výsledná funkce se sice v blízkosti uzlů chová celkem hezky, ale mezi uzly dochází ke kmitům, viz obr. 3.3, které jsou samozřejmě nežádoucí, ale jsou přímým důsledkem vysoké zaokrouhlené chyby ve výpočtech. Zde přichází na řadu metoda, které se česky říká splajn (z angl. spline). Opět využíváme funkce ve tvaru polynomu, ale tentokrát tak, že hledaná interpolační funkce je po celém intervalu rozdělená na malé části. Jednotlivé části mezi uzly jsou popsány polynomy nízkého stupně a v uzlech na sebe hladce navazují.

Pro představu si zde uvedeme pouze dva druhy splajnu, konkrétně Lineární a Kubický splajn. Existuje jich samozřejmě více a mají různé výhody i nevýhody, další informace najdete např. v doporučené literatuře.

3.3.1 Lineární interpolační splajn

Nejjednoduším druhem interpolace splajnem je lineární a neznamená nic jiného, než proložení dvou sousedních uzlů úsečkou. Mějme tedy dva sousední uzly dané souřadnicemi $[x_{i-1}, y_{i-1}]$ a $[x_i, y_i]$, které proložíme



Obrázek 3.3: Interpolace Rungeovy funkce pomocí Lagrangeova polynomu 10. stupně.

úsečkou, tedy lineárním interpolačním polynomem ve tvaru [7]

$$S_i(x) = y_{i-1} + \frac{(y_i - y_{i-1})}{(x_i - x_{i-1})}(x - x_{i-1}) = y_{i-1} + s\delta_i, \tag{3.9}$$

kde $S(x)$ je spojitá funkce, ale její první derivace ve vnitřních uzlech obecně spojitá není. Pokud známe dostatečné množství uzlů, pak lze chybu interpolace ovlivňovat a zmenšovat.

Úkol

Vysvětlete vlastními slovy, čím se od sebe liší lineární interpolace a lineární splajn.

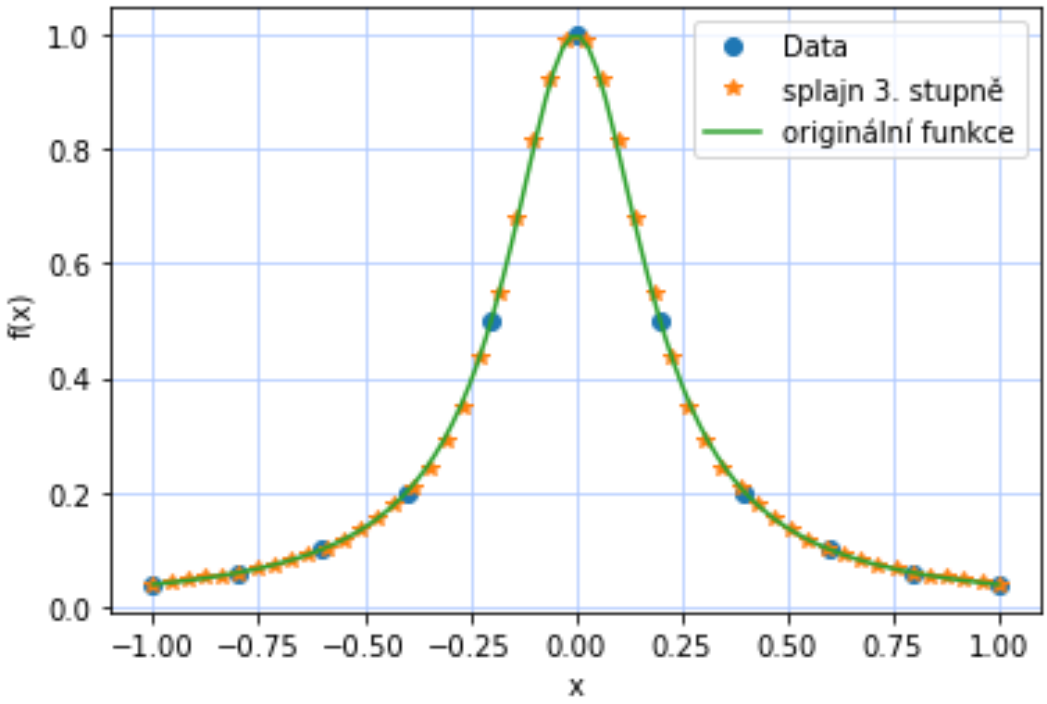
3.3.2 Kubický splajn

Velmi používanou aproximací je interpolace pomocí kubického splajnu. Využíváme tak vlastností polynomu 3. stupně, který je spojitý a má také spojité první i druhé derivace a zároveň má nízké riziko vzniku nežádoucích kmitů. Obr. 3.4 potvrzuje, že jsme pomocí kubického splajnu dostali naprosto vyhovující aproximaci Rungeovy funkce i při zachování počtu interpolačních uzlů.

Podrobnou definici zde pro jednoduchost uvádět nebudeme, ale krásně vysvětlený kubický splajn i s postupem řešení můžete najít na http://kfe.fjfi.cvut.cz/ psikal/nme/cv04/kubspline.pdf.

Úkol

Mějte zadanou funkcí pomocí 12ti interpolačních uzlů. Která z intepolačních metod bude nejvhodnější?



Obrázek 3.4: Interpolace Rungeovy funkce pomocí kubického splajnu.

3.4 Interpolace v programu `Python`

V programovacím jazyce `Python` existuje velká knihovna `Scipy`, která obsahuje velké množství různých druhů interpolací, jejich výčet lze najít na SciPy.org. Pro naše účely si výběrem jen některé druhy interpolací a jejich použití si ukážeme v následujícím příkladě.

Příklad

Nechť máme funkci $f(x_0) = \sin(x_0)$ a známe její funkční hodnotu v 5ti interpolačních uzlech, jejichž souřadnice jsou $x_0 = [0.0, 0.785, 1.571, 2.356, 3.142]$ na intervalu $< 0, \pi >$.

- Srovnejte, jakou výslednou hodnotu spočítají různé druhy interpolací dostupné v knihovně `Scipy` (konkrétně použijeme interpolaci lineárním polynomem, Lagrangeovu interpolaci, lineární splajn a také splajny 3. a 4. stupně), a to konkrétně v bodech $x = [0.2, 0.5, 0.9, 2.4]$ s hodnotami vypočítanými originální funkcí.
- Na základě získaných výsledků rozhodněte, která z interpolací je pro náš příklad nejvhodnější.

K výpočtu použijte níže uvedený kód v `Pythonu`.
Uvedeme zde řešení pouze pro $x = 0.2$ (ostatní jsou za úkol):

```
1  # Importujeme knihovny, které chceme použít
2  import numpy as np
3  from scipy import interpolate as inter
4  import math
5
6  # Poloha 5ti interpolačních uzlů a originální funkce, ze které dopočítáme hodnoty
7  x0 = np.array([0.0 , 0.785, 1.571, 2.356, 3.142])
```

```
y0 = [math.sin(i) for i in x0]

# Poloha bodů, ve kterých chceme znát interpolační hodnoty
x = np.array([0.2, 0.5, 0.9, 2.4])

# Naše interpolační funkce
funkce_linear = inter.interp1d(x0, y0, kind='linear')   # lineární interpolace
funkce_lagrange = inter.lagrange(x0, y0)                # Lagrangeův polynom
funkce_spline = inter.InterpolatedUnivariateSpline(x0, y0, k=1)         # splajn 1. stupně
funkce_cubic_spline = inter.InterpolatedUnivariateSpline(x0, y0, k=3)   # splajn 3. stupně
funkce_spline_4 = inter.InterpolatedUnivariateSpline(x0, y0, k=4) # splajn 4. stupně

print("Interpolovaná hodnota 'linear' v bodě", x[0], '=',  funkce_linear(x[0]))
print("Interpolovaná hodnota 'lagrange' v bodě", x[0], '=',  funkce_lagrange(x[0]))
print("Interpolovaná hodnota 'spline' v bodě", x[0], '=',  funkce_spline(x[0]))
print("Interpolovaná hodnota 'cubic spline' v bodě", x[0], '=',  funkce_cubic_spline(x[0]))
print("Interpolovaná hodnota 'cubic spline 4' v bodě", x[0], '=',  funkce_spline_4(x[0]))
print("Originální hodnota v bodě", 0.2, '=',  math.sin(0.2))
```

Program na obrazovku vypíše:
Interpolovaná hodnota 'linear' v bodě 0.2 = 0.18008284868926522
Interpolovaná hodnota 'lagrange' v bodě 0.2 = 0.19689743442955368
Interpolovaná hodnota 'spline' v bodě 0.2 = 0.18008284868926525
Interpolovaná hodnota 'cubic spline' v bodě 0.2 = 0.20516280705344686
Interpolovaná hodnota 'cubic spline 4' v bodě 0.2 = 0.19689743442955354
Originální hodnota v bodě 0.2 = 0.19866933079506122

Zkuste si najít nápovědu k funkci `interp1d` z knihovny `scipy.interpolate`, tato funkce je totiž hodně specifická a má velmi široký záběr. V našem příkladě ji používáme pouze k výpočtu lineární interpolace, ale v nápovědě najdete, že existují i volby `'nearest'`, `'nearest-up'`, `'zero'`, `'slinear'`, `'quadratic'`, `'cubic'`, `'previous'` a `'next'` a každá z nich počítá interpolaci odlišným způsobem.

Příklad

Nechť stále máme funkci $f(x_0) = \sin(x_0)$. Na intervalu $< 0, \pi >$ si pomocí příkazu v `Pythonu`

```
x0 = np.linspace(0, math.pi, num=10, endpoint=True)
```

napočítáme různý počet interpolačních uzlů (budeme měnit číslo `num`) v ekvidistantních vzdálenostech od sebe. Pomocí podobného příkazu

```
x = np.linspace(0, math.pi, num=100, endpoint=True)
```

si navolíme počet nových bodů, ve kterých chceme znát hodnotu interpolace. Vysledujte, jakým způsobem se mění předchozí výsledky, pokud zvyšuji nebo snižuji počet interpolačních uzlů. Diskutujte výsledky a použitelnost jednotlivých druhů interpolací.

Úkol

Existuje i jiný způsob, jak ověřit na kolik vybraná interpolace odpovídá původní funkci?

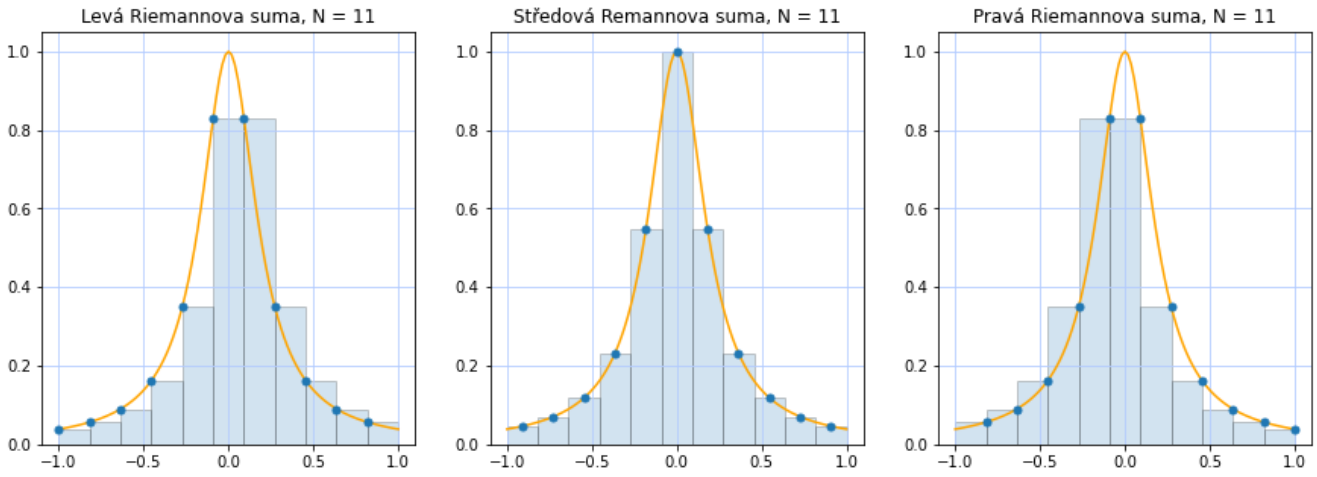
Příklad

Mějme Rungeovu funkci $g(x_0) = 1/(1 + 25x_0^2)$. Na intervalu $< -1, 1 >$ si pomocí příkazu v `Pythonu` spočítejte její integrační hodnotu. Poté funkci interpolujte všemi již uvedenými způsoby a spočítejte jejich integrační hodnoty. Výsledné hodnoty porovnejte a rozhodněte, která z interpolací je nejvhodnější.

```
# importované části knihoven
from math import sin, pi
from scipy.integrate import quad

# definice funkce, kterou chceme numericky integrovat
def g(x):
    return 1./(1.+25.*x**2.)

# použijeme vestavěnou funkci quad pro výpočet určitého integrálu od 0 do Pi
vysledek, chyba = quad(g, -1., 1.)
print("Numerický výsledek funkce je {:f} (+-{:g})" .format(vysledek, chyba))
```

Na obrazovku se nám vypíše:
Numerický výsledek funkce je 0.549360 (+-2.86683e-09).

Nyní si spočítáme, jaké integrační hodnoty dostaneme použitím různých interpolací, jinými slovy spočítáme obsahy ploch ohraničené interpolovanou křivkou. Ze znalostí o počítání určitých integrálů zvolíme Riemannův integrál. Na obr. 3.5 vidíme grafické srovnání levé, středové a pravé Riemannovy sumy spočítané pro případ Rungeovy funkce. Zvolili jsme dělení intervalu na 11 oddílů. Hned na první pohled vidíme jasné rozdíly ve způsobu výpočtu obsahů ploch pod křivkou.



Obrázek 3.5: Graficky znázorněný postup při počítání Riemannova integrálu Rungeovy funkce.

Lze samozřejmě použít i jiný druh integrálu, např. Simpsonovo pravidlo (více informací v knihovně `scipy.integrate.simpson`), které k výpočtu používá kvadratický polynom na každém oddílu křivky. Také

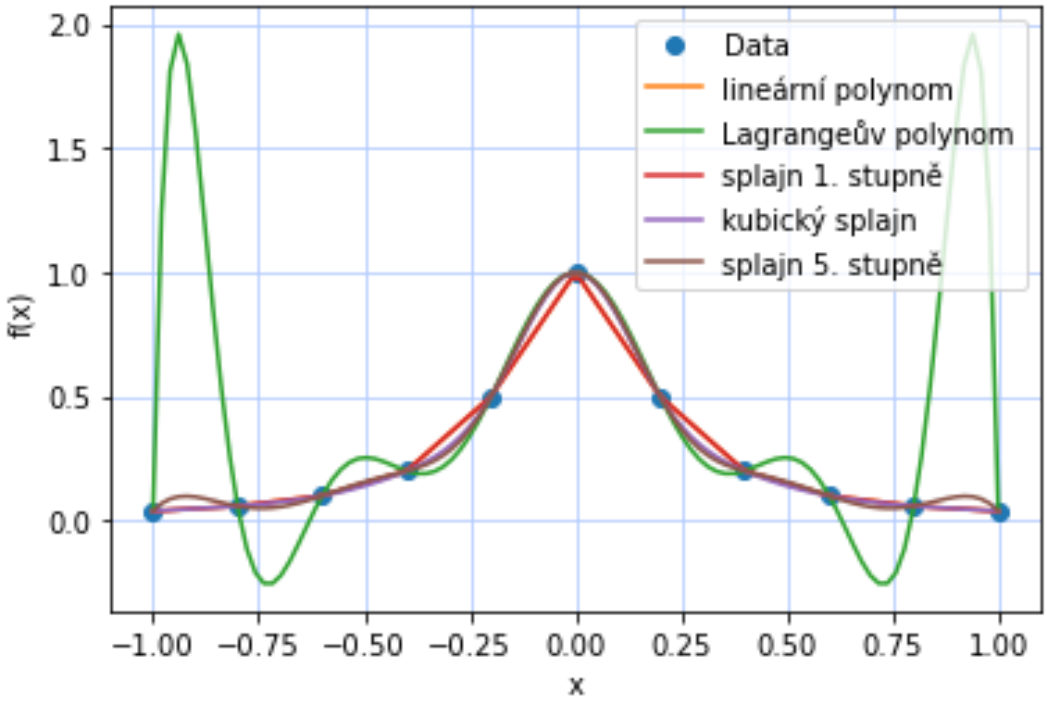
existuje lichoběžníkového pravidlo (`scipy.integrate.trapezoid`), které aproximuje každý oddíl křivky přímkou.

Z výše uvedeného kódu víme, jaká je přesná hodnota integrálu Rungeovy funkce na intervalu $< -1, 1 >$. Použitím Riemannovy sumy dostáváme hodnoty (pro případ dělení na 11 oddílů):
Levá Riemannova suma: 0.5477044454139318,
Středová Riemannova suma: 0.4956374418136541,
Pravá Riemannova suma: 0.33904407478069454.

Pro znázornění výpočtu jsme použili interpolační funkci nazvanou `funkce_spline`, která nám interpoluje Rungeovu funkci pomocí splajnu 1. stupně. Výpočty ostatních druhů interpolací jsou za úkol.

```
# Importujeme knihovny, které chceme použít
import numpy as np
import matplotlib.pyplot as plt
from scipy import interpolate as inter

# Poloha 11ti interpolačních uzlů a originální funkce, ze které dopočítáme hodnoty
x0 = np.linspace(-1, 1, num=11, endpoint=True)
y0 = [1/(1.+(25.*(i)**2)) for i in x0]

# Poloha bodů, ve kterých chceme znát interpolační hodnoty
x = np.linspace(-1, 1, num=100, endpoint=True)

# Naše interpolační funkce
funkce_linear = inter.interp1d(x0, y0, kind='linear', bounds_error=False, fill_value='extrapolate')
funkce_lagrange = inter.lagrange(x0, y0)                # Lagrangeův polynom
funkce_spline = inter.InterpolatedUnivariateSpline(x0, y0, k=1)         # splajn 1. stupně
funkce_cubic_spline = inter.InterpolatedUnivariateSpline(x0, y0, k=3)   # splajn 3. stupně
funkce_spline_5 = inter.InterpolatedUnivariateSpline(x0, y0, k=5) # splajn 5. stupně

# Do grafu vykreslíme modrou gridovací mřížku
plt.plot(x0, y0, 'o', label='Data')
plt.grid(linestyle="-", color=(0.7, 0.8, 1.0))

# Vykreslíme si všechny použité interpolace do jednoho grafu
plt.plot(x, funkce_linear(x), '-', label='lineární polynom')
plt.plot(x, funkce_lagrange(x), '-', label='Lagrangeův polynom')
plt.plot(x, funkce_spline(x), '-', label='splajn 1. stupně')
plt.plot(x, funkce_cubic_spline(x), '-', label='kubický splajn')
plt.plot(x, funkce_spline_5(x), '-', label='splajn 5. stupně')

# Označíme osy v grafu
plt.xlabel('x')
plt.ylabel('f(x)')
# Legendu v grafu
plt.legend()

plt.show()

# Předpisy pro výpočet Riemannovy sumy ve 3 různých polohách
a = -1
```

```
b = 1
N = 11
dx = (b-a)/N
x_vlevo = np.linspace(a, b - dx, N)
x_uprostred = np.linspace(dx/2,b - dx/2,N)
x_vpravo = np.linspace(dx,b,N)

# Výpočet Riemannova integrálů pro různé polohy
Riemannova_suma_vlevo = np.sum(funkce_spline(x_vlevo) * dx)
Riemannova_suma_uprostred = np.sum(funkce_spline(x_uprostred) * dx)
Riemannova_suma_vpravo = np.sum(funkce_spline(x_vpravo) * dx)

# Tisk výsledků
print("Levý Riemannův integrál:", Riemannova_suma_vlevo)
print("Středový Riemannův integrál:", Riemannova_suma_uprostred)
print("Pravý Riemannův integrál:", Riemannova_suma_vpravo)
```



Obrázek 3.6: Různé druhy interpolací aplikovaných na Rungeovu funkci.

Spuštěním výše uvedeného kódu obdržíme obr. 3.6 a text:
Levý Riemannův integrál: 0.5473018959649976
Středový Riemannův integrál: 0.5027747653416103
Pravý Riemannův integrál: 0.35668449197860963

Porovnáme nyní získané hodnoty integrálů Rungeovy funkce:
`quad` funkce: $0.549360(+-2.86683e-09)$
Riemannova levá suma aplikovaná na originální funkci: 0.5477044454139318
Levý Riemannův integrál z `funkce_spline` : 0.5473018959649976

Všechny získané výsledky takto porovnejte a diskutujte.

Úkoly

1. Jak můžeme na obr. 3.6 vidět, tak např. interpolace splajnem 5. stupně už nebude ideální, zkuste si spočítat její Riemannův intergrál a opět diskutujte výsledky.
2. Zkuste změnit počet oddílů, na které se plocha pod křivkou dělí, jak moc to ovlivní výsledky?
3. Vyměňte Rungeovu funkci např. za funkci $f(x_0) = \sin(x_0)$ a zkuste spočítat výše uvedené Riemannovy integrály.
   i) Který z Riemannových integrálů bude nejblíže odpovídat hodnotě originální funkce?
   ii) Zkuste změnit i počet oddílů, na které si plochu pod křivkou rozdělíte. Diskutujte výsledky.

Souhrn kapitoly

V této kapitole jste se dozvěděli, k čemu a jak se používá interpolace a extrapolace. Vysvětlili jsme Vám různé druhy interpolací, popsali jsme, kde a jak se dají využít a pro jaké případy jsou či nejsou vhodné. Pomocí programů psaných v programovacím jazyce `Python`, využitím hlavně knihovny `Scipy` a jejich vestavěných funkcí, jste měli za úkol spočítat interpolační hodnoty v různých bodech křivky za použití různých interpolačních metod. Naučili jste se, jakým způsobem ověřit kvalitu Vámi zvolené interpolační metody i za použití Riemannova integrálu.

Další informace

- PRESS William H., TEUKOLSKY Saul A., VETTERLING William T., FLANNERY Brian P. Numerical recipes. The art of scientific computing. Third edition. *Cambridge*. 2007, ISBN-13 978-0-521-88068-8
- ČERMÁK, Libor a Rudolf HLAVIČKA. Numerické metody I. In: *VUT, FSI, Ústav matematiky* [online]. Brno, 1.11.2006. Dostupné z: http://mathonline.fme.vutbr.cz/Numericke-metody-I/sc-8-sr-1-a-11/default.aspx

Kapitola 4

Numerická řešení rovnic, hledání kořenů

Rychlý náhled: Tato kapitola se zabývá hledáním kořenů funkcí. Ukážeme si dvě základní metody a to metodu půlení intervalu (bisekci) a Newtonovu Raphsonovu metodu (metodu tečen). Metody budeme nejprve aplikovat na hledání kořenů kvadratické funkce. V druhé části si ukážeme aplikaci na hledání rovnovážných bodů gravitačně vázaných systémů.

Cíle studia:

- Pochopit matematické pozadí základních numerických metod pro hledání kořenů funkcí.
- Umět používat metody hledání kořenů, které jsou součástí knihovny `scipy`.
- Seznámit se s Rocheovým potenciálem a jeho vlastnostmi.

Všechny rovnice lze převézt na tvar $f(x) = 0$ a proto se budeme v této kapitole zabývat pouze rovnicemi tohoto typu. Úloha tedy spočívá v nalezení x_0 pro které platí $f(x_0) = 0$. Hledání řešení této rovnice se často označuje jako hledání kořenů (root finding). Pro řadu případů může být takových řešení více. Např. kvadratická rovnice $x^2 - 4 = 0$ má tato řešení dvě, a to $x_1 = 2$ a $x_2 = -2$. Některé úlohy naproti tomu řešení v oboru reálných čísel nemají, např. rovnice $x^2 + 4 = 0$.

Při řešení rovnice numericky je proto třeba znát o řešení nějaké informace (např. řešení existuje přibližně zde, nebo řešení existuje na intervalu (a,b), který známe). Toto se může zdát jako značné omezení nicméně při řešení reálných problémů v přírodních vědách takové informace často máme k dispozici.

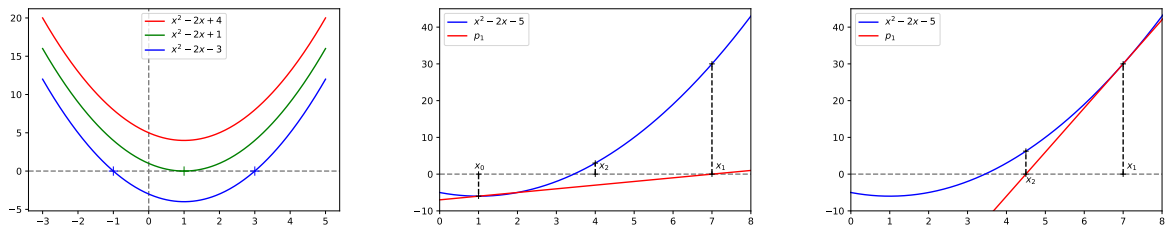
My si nejdříve ukážeme numerické hledání kořenů kvadratické funkce[1] a posléze se zaměří na složitější funkci, které je ovšem zajímavá z hlediska výzkumu Vesmíru, konkrétně dvojhvězd.

4.1 Hledání kořenů kvadratické funkce

Začněme zopakováním základních vlastností kvadratické rovnice. Mějme kvadratickou rovnici

$$x^2 + ax + b = 0, \tag{4.1}$$

[1] Z terminologického hlediska je hledání kořenů funkce $f(x)$ ekvivalentní s řešením rovnice $f(x) = 0$. Vždy ale hovoříme o řešení (řešeních) rovnice, případně o kořenu (kořenech) funkce.



Obrázek 4.1: *Vlevo:* Znázornění vlivu koeficientu b na kvadratickou funkci. Znázorněny jsou situace, kdy existují dva kořeny (modře), jeden kořen (zeleně) a situace kdy kořen neexistuje (červeně). *Uprostřed:* Znázornění metody půlení intervalů. *Vpravo:* Znázornění Newtonovy Raphsonovy metody.

kde uvažujeme, že člen u x^2 je roven jedné. Pokud by byl člen u x^2 různý od jedničky, můžeme celou rovnici vydělit tímto koeficientem a převézt ji na tvar (4.1).

Z hlediska hledání řešení kvadratické rovnice mohou nastat tři situace v závislosti na hodnotě tzv. diskriminantu, který v případě rovnice ve tvaru (4.1) vypadá $D = a^2 - 4b$

- $D < 0$ rovnice (4.1) řešení nemá.
- $D = 0$ tj. $b = a^2/4$ má rovnice jedno řešení $x_0 = -a/2$ a rovnici lze přepsat na tvar $(x + a/2)^2 = 0$.
- $D > 0$ má rovnice dvě řešení $x_1 = -a/2 + \sqrt{D}/2$ a $x_2 = -a/2 - \sqrt{D}/2$.

Podíváme-li se podrobně na výše uvedené vidíme, že pokud kořeny kvadratické funkce existují leží jeden z nich vlevo od $-a/2$ a druhý leží vpravo od $a/2$. K podobnému závěru bychom došli, pokud bychom využili známých vztahů koeficientů a a b ke kořenů rovnice x_1 a x_2

$$a = -(x_1 + x_2), \tag{4.2}$$
$$b = x_1 x_2. \tag{4.3}$$

I z těchto vlastností vyplývá, že koeficient $-a/2$ je možné brát jako průměr koeficientů x_1 a x_2 a že intervaly na kterých se nachází právě jeden kořen jsou tedy $(-\infty, -a/2)$ a $(-a/2, \infty)$. Ke stejnému závěru jsme došli již dříve ze znalosti kořenů.

4.1.1 Hledání kořenů metodou půlení intervalů

Metoda půlení intervalů je jednou ze základních metod hledání kořenů. Postup vychází z počáteční znalosti intervalu na kterém se kořen nachází. V ukázce budeme hledat větší kořen tj. kořen který se nachází v intervalu $(-a/2, \infty)$. Nekonečno je z pochopitelných důvodů jako okrajový bod nevhodné, proto zkusíme najít vhodnější bod. Uděláme odhad, že tímto bodem by mohl být bod $x' = a/2$ a spočítáme hodnotu levé strany rovnice v tomto bodě. Vzhledem k naší volbě koeficientu u x^2 víme, že hodnota levé strany rovnice v $x_0 = -a/2$ musí být záporná, aby kořeny existovaly[2]. Pokud je hodnota v $x' = a/2$ kladná, máme vyhráno a víme, že kořen leží na intervalu (x, x'). Pokud je hodnota levé strany rovnice v bodě x' záporná stejně jako v bodě x_0 proložíme těmito body přímku a nalezneme průsečík x_1 této přímky z osou x. Vzhledem k tomu, že přímka leží na intervalu (x', ∞) pod parabolou musí i v bodě, kde přímka protíná osu x ležet

[2] Je-li koeficient u $x^2 > 0$ je parabola obrácená vrcholem dolu (tj. v bodě $x_0 = -a/2$ leží minimum kvadratické funkce). Nachází-li se toto minimum na osou x, průsečíky s osou x neexistují a tím pádem nelze kořeny nalézt.

parabola nad ní a hodnota musí levé strany rovnice musí být v bode x_1 kladná. Nyní tedy již bezpečně známe interval na kterém kořen leží, označme ho $(x0, x1)$.

Při zmenšování intervalu postupujeme následovně. Levou stranu rovnice si označíme jako $f(x)$ a levé strana rovnice je rovna nula. Zvolíme $x_2 = (x_1 - x_0)/2$. Spočítáme $f(x_2)$. Je-li $f(x_2) > 0$ víme, že kořen leží na intervalu (x_0, x_2). Je-li $f(x_2) < 0$ víme, že kořen leží na intervalu (x_2, x_1). Nový interval označíme (x_0, x_1) spočítáme nové x_2 a postupuje do doby než je splněna podmínka $x_1 - x_0/(x_1 + x_0) < \eta$, kde η značí námi požadovanou přesnost. Řešení potom můžeme psát jako $x = (x_0 + x_1)/2$.

Příklad

Následující část ukazuje využití metody půlení intervalů v pythonu.

```
1 from scipy import optimize as opt
2
3 def f(x):
4     return x**2-2*x - 5
5
6 koren=opt.bisect(f,1,7)
7
8 print(koren)
```

Program vypíše hodnotu 3.4494897427834985. Na řádku 1 importujeme z knihovny `scipy` rutinu `optimize`. Na řádku 3 definujeme kvadratickou funkci. Hledaný `koren` potom najdeme metodou `optimize.bisect`, kde prvním argumentem je funkce jejíž kořen hledáme, druhým a třetí argument určují interval na kterém se kořen hledá – v našem případě se jedná o interval $(1, 7)$.

Úkol

Vyřešte následující úkoly.

1. Výše uvedený postup jsme aplikovali na hledání kořenu vpravo od minima kvadratické funkce. Jak by se postup změnil pro hledání kořenu vlevo od minima? Jak byste poznali zda hledaný kořen leží v intervalu (x_0, x_2) či (x_2, x_1) v obecném případě, kdy nevíte zda funkce $f(x)$ na intervalu (x_0, x_1) roste či klesá.

2. Pročtěte si dokumentaci `scipy.optimize.bisect`

3. Jakým způsobem můžeme nastavit přesnost s jakou se kořen hledá? Nastavte přesnost na různé hodnoty a porovnejte výsledky.

4. Nastavte přesnost na poměrně vysokou hodnotu (např. `xtol=0.01` a zkuste změnit pravý okraj intervalu na kterém kořen hledáme. Má toto nastavení vliv na výsledek? Je výsledek vždy v požadované toleranci od přesného řešení, které získáte přímým výpočtem?

5. Upravte jednoduchý skript tak, aby našel druhý kořen kvadratické rovnice.

4.1.2 Hledání kořenů Newtonovou – Raphsonovou metodou

Metoda půlení intervalů je vhodná pro úlohy kde známe interval na kterém se kořen nachází. Někdy můžeme hledat kořeny funkce u které víme, kde přibližně kořen leží. Toho poté můžeme využít pro najití kořenu s libovolnou přesností.

Metoda je znázorněna vlevo na Obr.4.1. Vyjdeme z prvotního odhadu místa, kde se kořen nachází. V případě kvadratické funkce $f(x) = x^2 - 2x - 5$ zvolíme bod vpravo od vrcholu paraboly ($x_0 = 1$). V ukázce jsme zvolili $x_1 = 7$. V bodě x_1 nalezneme funkční hodnotu ($f(x_1)$) a numericky spočítáme derivaci v tomto bodě. Z znalosti derivace spočítáme tečnou přímku v bodě x_1. Na obrázku je znázorněna červeně a označena jako p_1. Spočítáme průsečík p_1 s osou x a průsečík použijeme jako nový odhad kořenu x_2. Pokračujeme dále iterativně dokud nenalezneme kořen s požadovanou přesností.

Příklad

Následující část ukazuje využití Newtonovy Raphsonovy metody v pythonu.

```
1 from scipy import optimize as opt
2
3 def f(x):
4     return x**2-2*x - 5
5
6 koren=opt.newton(f,7)
7
8 print(koren)
```

Program vypíše hodnotu 3.449489742783178. Na řádku 1 importujeme z knihovny `scipy` rutinu `optimize`. Na řádku 3 definujeme kvadratickou funkci. Hledaný `koren` potom najdeme metodou `optimize.newton`, kde prvním argumentem je funkce jejíž kořen hledáme, druhým argumentem je prvotní odhad na kořen, v našem případě jsme zvolili $x_1 = 7$. .

Úkol

Vyřešte následující úkoly.

1. Pročtěte si dokumentaci `scipy.optimize.newton`
2. Jakým způsobem můžeme nastavit přesnost s jakou se kořen hledá? Nastavte přesnost na různé hodnoty a porovnejte výsledky.
3. Upravte jednoduchý skript tak, aby našel druhý kořen kvadratické rovnice.

Další informace

- Podívejte se na další metody hledání kořenů funkcí. Při řešení vlastních problémů vždy dbejte o to, abyste měli odhad jaké řešení hledáte. Např. si můžete vykreslete funkci, jejíž kořen hledáte apod. Pokud používáte jiné programovací jazyky než python můžete využít např. numerical recipes nebo GSL

- Dokumentaci k probíraným knihovnám naleznete online pro půlení intervalů i pro Newtonovu Raphsonovu metodu

4.2 Pohyb částic v poli dvou kolem sebe obíhajících objektů

Zabývejme se situací kdy máme dvě gravitačně vázaná tělesa, která obíhají po kruhových trajektoriích kolem vzájemného těžiště. Nás zajímá pohyb mnohem lehčího tělesa v jejich blízkosti. Taková situace je například soustava Slunce a Jupiter, které obíhají kolem společného těžiště a nás zajímá pohyb asteroidů, které jsou gravitačně ovlivňovány oběma tělesy.

Z hlediska výpočetní náročnosti je vhodné zvolit souřadnou soustavu ve které je Slunce i Jupiter v klidu a výpočet provádět v ní. Asteroid, který je v klidu v této soustavě ve skutečnosti obíhá kolem středu soustavy stejnou úhlovou rychlostí jako Jupiter.

4.2.1 Teorie

Uvažujeme dvě tělesa o hmotnostech M_1 a M_2 ve vzdálenosti a, kde jsme hmotnější těleso označili indexem 1 a platí tedy $M_1 > M_2$. My se omezíme na jednoduchý příklad, kdy nás budou zajímat drobná tělesa, která se pohybují ve stejné rovině jako vzájemně se obíhající tělesa. V případe soustavy Slunce – Země by se jednalo o tělesa která se pohybují v rovině ekliptiky. My si tuto rovinu označíme jako rovinu x, y. Tělesa umístíme na osu x a do počátku soustavy souřadnic umístíme hmotný střed.

Je-li hmotný střed v bodě $[0, 0]$ pak pro souřadnice těles musí platit

$$M_1 r_1 = M_2 r_2, \tag{4.4}$$

kde r_i je vzdálenost daného tělesa od hmotného středu. Zároveň známe vzdálenost obou těles a, tedy platí $r_1 + r_2 = a$. Hmotnější těleso má tedy souřadnice $[x_1, 0]$, těleso méně hmotné má souřadnice $[x_2, 0]$, kde $x_{1,2}$ jsou dány následovně

$$\begin{aligned} x_1 &= -r_1 = -\frac{M_2 a}{M_1 + M_2}, \\ x_2 &= r_2 = \frac{M_1 a}{M_1 + M_2}. \end{aligned} \tag{4.5}$$

Úhlová rychlost, kterou tělesa obíhají kolem společného hmotného středu je dána Keplerovým zákonem

$$\omega^2 = G\frac{M_1 + M_2}{a^3}, \tag{4.6}$$

kde $G = 6.6724 \times 10^{-11}$ je gravitační konstanta[3].

Pohybové rovnice v námi uvažované soustavě je možné zapsat ve tvaru

$$\begin{aligned} \ddot{x} &= -\frac{\partial U_{\mathrm{R}}}{\partial x} + 2\omega v_y, \\ \ddot{y} &= -\frac{\partial U_{\mathrm{R}}}{\partial y} - 2\omega v_z, \end{aligned} \tag{4.7}$$

kde U_{R} je Rocheův potenciál definovaný následujícím způsobem

$$U_{\mathrm{R}}(x, y) = -\frac{GM_1}{d_1} - \frac{GM_2}{d_2} - \frac{1}{2}\omega^2(x^2 + y^2), \tag{4.8}$$

[3] Hodnotu gravitační konstanty uvádíme dle aktuálních měření ze stránek NIST.

kde $d_{1,2}$ jsou vzdálenosti bodu o souřadnicích (x, y) od těles 1, 2 a můžeme je psát následujícím způsobem

$$d_1^2 = (x - x_1)^2 + y^2, \tag{4.9}$$
$$d_2^2 = (x - x_2)^2 + y^2. \tag{4.10}$$

V Rocheově potenciálu (4.8) mají první dva členy význam gravitačních potenciálů polí generovaných tělesy o hmotnostech M_1, M_2. Třetí člen odpovídá potenciálu odstředivé síly, která působí v rotující soustavě i na částici která je v této soustavě v klidu.

V rovnicích (4.7) odpovídá druhý člen příslušné složce Coriolisova zrychlení

$$a_{\mathrm{Cor}} = -2\vec{\omega} \times \vec{v}. \tag{4.11}$$

V našem případě je $\vec{\omega} = (0, 0, \omega)$ a $\vec{v} = (v_x, v_y, 0)$. Coriolisovo zrychlení nepůsobí na částice, které jsou v soustavě v klidu, ale působí na pohybujících se částice.

Rocheův potenciál graficky znázorňujeme na Obr.4.2. Potenciál má pět lokálních extrémů, které značíme L_i a nazýváme je Lagrangeovými body. V těchto bodech je $\partial U_{\mathrm{R}}/\partial x = \partial U_{\mathrm{R}}/\partial y = 0$. Bod L_1 leží mezi tělesy, body L_2 a L_3 leží na ose x vně soustavy. Body L_4 a L_5 leží ve vrcholech rovnostranných trojúhelníků se základnou danou tělesy M_1 a M_2.

Body L_1, L_2 a L_3 jsou nestabilní a částice v nich umístěná by vlivem malé perturbace uletěla nakonec pryč. Body L_4 a L_5 jsou stabilní pokud je[4]

$$M_1 > 25M_2 \left(\frac{1 + \sqrt{1 - 4/625}}{2} \right). \tag{4.12}$$

K tomu dochází např. v soustavě Slunce – Jupiter, kde se v bodech L_4 a L_5 pohybují tzv. Trójské asteroidy. Naproti tomu v binárních systémech dvou hvězd tato situace většinou nenastává.

4.2.2 Implementace v Pythonu

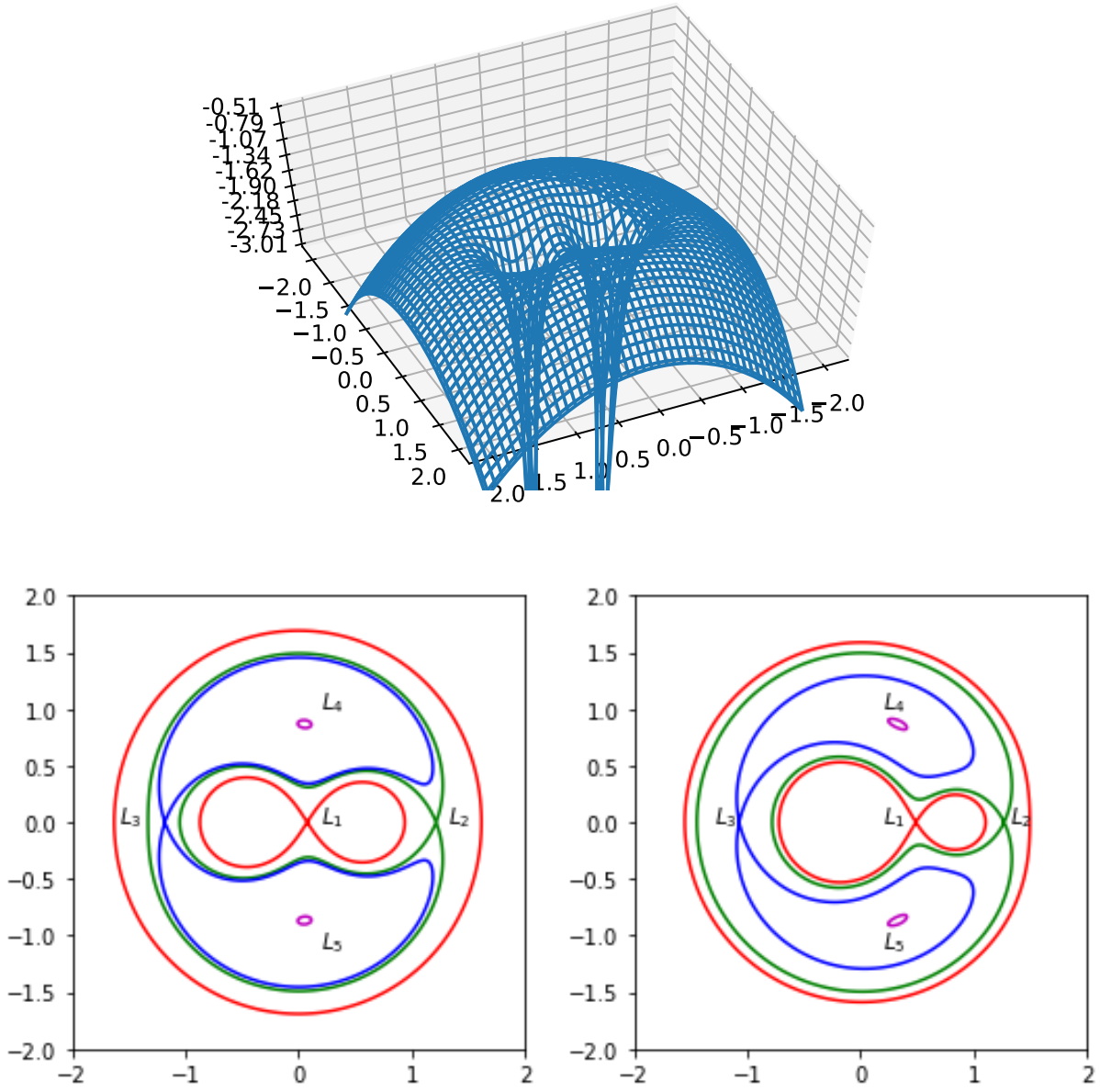
V repozitáři na githubu si můžete stáhnout skript, který řeší pohyb asteroidu v blízkosti bodu L_4 soustavy Slunce – Jupiter. My se zde zaměříme na hledání bodů L_1, L_2 a L_3 pomocí Newtonovy - Raphsonovy metody.

Začneme definováním základních veličin

```
DistanceJ = 778570000000. # m JUPITER FROM SUN
G = 6.6724*10**-11
Jupiter_mass = 1.8982*10**27 # kg
Sun_mass = 1.989*10**30 # kg


M1=Sun_mass
M2=Jupiter_mass
a=DistanceJ
Ang_vel=math.sqrt(G*(M1+M2)/(a**3))
r2=M1*a/(M1+M2)
r1=M2*a/(M1+M2)
```

Nadefinujeme Rocheův potenciál $U_{\mathrm{R}}(x, y, v_x, v_y)$

[4] Zájemci o odvození se mohou podívat např. zde.



Obrázek 4.2: Znázornění Rocheova potenciálu. Souřadnice na osách x a y jsou v jednotkách a.*Nahoře:* Hodnoty Rocheova potenciálu (v jednotkách $\omega^2 a^2$ pro $M_2 = 0.8\,M_1$. *Vlevo dole:* Ekvipotenciály pro hodnoty, které odpovídají Lagrangeovým bodům. Ty jsou označeny znázorněny obrázku jako $L_1 ... L_5$. I zde je $M_2 = 0.8\,M_1$. *Vpravo dole:* Stejné jako vlevo dole s tím, že zde je $M_2 = 0.2\,M_1$

```
def pot(x,y):
    x1=-r1
    x2=r2
    d1=math.sqrt((x-x1)**2 + y**2 )
    d2=math.sqrt((x-x2)**2 + y**2 )
    result = -G*(M1/d1 + M2/d2) -1.*Ang_vel*Ang_vel*(x**2 + y**2)/2.
    return result
```

Zrychlení ve směru osy x $a_x = \partial U_{\mathrm{R}}(x, y, v_x, v_y)/\partial x$ je definováno ve funkci acx, zatímco $acx_a x$ je speciální případ zrychlení působící na částici v klidu ($v_x = v_y = 0$) na ose x ($y = 0$). My zde implementujeme vlastní derivaci, ale mohli bychom využít i předdefinované (např. `derivative` z knihovny `scipy.misc`).

```
def acx(x,y,vx,vy):
    dx=a/1000.
    result=-(pot(x+dx,y) - pot(x-dx,y))/(2.*dx)    + 2.* Ang_vel*vy
#    result=-(-pot(x+2.*dx,y) + 8.*pot(x+dx,y) - 8.*pot(x-dx,y) + pot(x-2.*dx,y))/(12.*dx)
#                     + 2.* Ang_vel*vy
    return result

def acx_ax(x):
    return acx(x,0,0,0)
```

Hledání extrémů potenciálu U_{R} na ose x můžeme přepsat na hledání kořenů funkce $a_x(x)$. Tedy hledáme místa, kde na umístěnou částici nepůsobí žádná síla.

Definujme parametr $\alpha = M_2/(M_1 + M_2)$. Je-li tento parametr malý $\alpha \ll 1$ potom je možné vyjádřit pozici bodů $L_i[x_i, 0]$ vztahy

$$x_1 = a\left[1 - \left(\frac{\alpha}{3}\right)^{1/3}\right] \tag{4.13}$$

$$x_2 = a\left[1 + \left(\frac{\alpha}{3}\right)^{1/3}\right] \tag{4.14}$$

$$x_3 = -a\left[1 + a\frac{5}{12}\alpha\right] \tag{4.15}$$

. Tyto body využijeme jako odhad kořenu pro Newtonovu Raphsonovu metodu.

```
alpha=M2/(M1+M2)
P0=a*a*Ang_vel*Ang_vel

L1X_0=a*(1.-(alpha/3.)**(1./3.))
L1X=opt.newton(acx_ax,L1X_0,tol=a*10**-5)
L1Y=0.
P1=pot(L1X,L1Y)/P0

print(L1X_0/a,L1X/a)
```

V uvedené funkci je `P0` jednotka, která se používá pro měření potenciálu. `L1X_0` značí původní odhad a `L1X` přesnější řešení. Jako požadovanou přesnost jsme uvedli $10^{-5}a$. Je důležité vědět s jak velké hodnoty kořen má a nastavit přesnost na rozumnou hodnotu. Pro soustavu Slunce – Jupiter dostaneme

`L1X_0=0.931757139893148` a `L1X=0.9323711678782919`. Zkusíme si nyní nastavit hodnotu $M_2 = 0.1M_0$. Dostáváme `L1X_0=0.688234046118128` a `L1X=0.6266027208867219`. Vidíme, že nyní je přesná hodnota od původní dále, což jsme očekávali, neboť poměr hmotností byl nyní menší.

Úkol

Vyřešte následující úkoly.

1. Najděte polohy dalších Lagrangeových bodů.
2. Prozkoumejte chování pro další poměry hmotností.
3. Upravte kód pro soustavu Slunce – Země a pro soustavu Země – Měsíc.

Další informace

- Stáhněte si kód z Github a zkuste si vykreslit trajektorie pro různé kombinace hmotností a různé počáteční polohy asteroidů.

Souhrn kapitoly

V této kapitole jsme se zabývali metodami hledání kořenu funkcí. Newtonovu Raphsonovu metodu jsme poté aplikovali na hledání rovnovážných poloh v poli dvou kolem sebe obíhajících objektů.

Kapitola 5

Statistický popis dat, distribuční funkce

Adam Hofer

Rychlý náhled: Tato kapitola se zabývá statistikou, jež nám slouží k popisu libovolného datového souboru. Jedná se jak o popis dat za pomoci charakteristik, což je popsáno základní deskriptivní statistikou. Dále jsou zde popsány nejčastěji používané distribuční funkce

Klíčová slova: statistika, popis dat, distribuční funkce, normální rozdělení, PDF, CDF, variabilita, průměr

Cíle studia: V této kapitole je cílem studenta naučit pracovat se základními statistickými metodami pro popis dat. Po přečtení tak bude student schopen říct základní informace o libovolných datech a popsat je za pomocí základních charakteristik. Dále budete schopní v `Pythonu` naprogramovat jednoduché funkce pro zpracování dat a jejich základní statistický popis.

Pokud chceme dobře porozumět světu okolo nás, je vhodné znát alespoň základní statistické metody. Proto si v této kapitole vybrané metody popíšeme. Jelikož naše okolí lze často popisovat za pomoci různých měření a následně tak naměřenými daty. Ty však mohou být poměrně rozsáhlé, tudíž je vhodné začít s první částí statistiky a tou je statistika popisná neboli deskriptivní. Ta se nám naměřený soubor dat snaží popisovat pomocí několika číselných hodnot, jež o daných datech něco vypovídají. Příkladem může být měření teploty v průběhu dne. Dejme tomu, že budeme měřit teplota každou čtvrthodinu. Za jeden den tak získáme 96 hodnot. Z těchto hodnot máme velmi podrobnou představu o počasí v daném dni. Nicméně když budeme chtít někomu dalšímu popsat jaký bylo ten daný den počasí sdělovat mu všech 96 hodnot by bylo jistě nadbytečné. Proto si vystačíme s několika charakteristikami jež nám ten soubor teplot popíšou. Může to být například minimum, maximum a průměr. K získaní představy o počasí v daném dni mi tak stačí tyto tři hodnoty a to je v základu princip popisné statistiky. Samozřejmě charakteristik existuje vícero než v tomto našem jednoduchém příkladě. Proto se s nimi pojďme seznámit:

5.1 Základní charakteristiky popisné statistiky

Tyto charakteristiky jsou nám schopny dobře popsat datový soubor a říct nám základní informace o obsažených datech.

5.1.1 Minimum

nejnižší hodnota v datovém souboru. Pro její nalezení nám v `Pythonu` slouží funkce `min(list)`

5.1.2 Maximum

nejvyšší hodnota v datovém souboru, kterou můžeme nalézt použitím funkce `max(list)`

5.1.3 Počet hodnot

Neboli celkový počet vzorků v datovém souboru. Tento údaj o počtu hodnot v datovém souboru je nezbytný pro většinou dalších výpočtů. V `Pythonu` jej můžeme najít funkcí `len(list)`, kterou aplikujeme na datový soubor.

5.1.4 Aritmetický průměr

Základní charakteristika popisující střed dat. Anglicky jej nazýváme *mean* a označujeme jej $\bar{x}$. Spočteme jej tak, že sečteme veškeré hodnoty v datovém souboru a tento součet vydělíme počtem hodnot. Tato charakteristika je nicméně náchylná na odlehlé hodnoty, které nám mohou výsledný průměr zkreslovat. Příkladem budiž platy. Například máme 9 dělníku pracujících za minimální mzdu a jednoto top manažera s milionovým příjmem. Pokud spočteme aritmetický průměr jejich platů získáme částku která rozhodně nepopisuje reálný příjem většiny zaměstnanců v dané firmě, ale je zkreslena odlehlou hodnotou platu manažera. V `Pythonu` můžeme (díky `statistic module`) aritmetický průměr spočíst funkcí `mean(list)`. Případně můžeme využít knihovnu `Numpy` a funkci `numpy.mean()`.

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{5.1}$$

Úkol

Napište vlastní funkci pro výpočet aritmetického průměru. Takže nám ale nic nebrání napsat si vlastní funkci za pomoci `sum()` a `length()`. Funkcí sum prvně sečteme položky dle předchozího vzorce a následně vydělíme počtem prvků o což se nám postará funkce `length`.

5.1.5 Medián

Je charakteristika středu a je to hodnota u níž platí, že 50 % dat je větších nebo rovno než medián a 50 % dat je menších nebo rovno. Při sudém počtu vzorků pak spočítáme aritmetický průměr dvou prostředních čísel. V `Pythonu` jej spočteme funkcí `median(list)`. Označujeme jej $\tilde{x}$.

5.1.6 Modus

Další z charakteristik, kdy spočítáme počty výskytů jednotlivých prvků a modus je ten nejčastější. V `Pythonu` se funkce nazývá `mode(list)`. Označujeme jej $\hat{x}$.

5.1.7 Kvantily

Jde o hodnoty, které nám rozdělují setříděná data podobně jako medián. Medián je tedy speciální případ kvantilů. Tedy kvantil $x_0.33$ rozděluje data tak, že 33 % dat je menších nebo rovno než $x_0.33$a 66 % je větších nebo rovno než $x_0.33$. V `Pythonu` jej díky knihovně `numpy` můžeme spočítat funkcí `numpy.quantile()`.

5.1.8 Rozpětí

Základní a nejjednodušší míra variability. Získáme jej tak, že stačí odečíst minimum a maximum. Na data bez extrému (outliers) to stačí, ale co třeba na $(1, 2, 3, 4, 500)$? Rozpětí vychází $500 - 1 = 499$, přitom většina dat je z rozsahu 1 až 4.

5.1.9 Mezikvartilové rozpětí a odchylka

Řeší problémy s extrémy. Budeme pracovat s rozdíly kvartilů na rozdíl od klasického rozpětí. Pokud budeme chtít popsat, jak se data odchylují od mediánu, vydělíme jej dvěma.

5.1.10 Rozptyl

Nejpoužívanější míra rozptýlenosti. Spočteme jej tak, že sečteme druhé mocniny všech odchylek od průměru. Jde o celkovou míru rozptýlenosti ta nám ale moc neříká, jak jsou data průměrné rozptýlená. Označujeme jej σ^2 či s^2 či $Var(X)$. Lze jej spočíst za pomoci následujícího vzorce

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 \tag{5.2}$$

Pro výpočet za pomoci `Pythonu` můžeme použít funkci `numpy.var()`.

5.1.11 Směrodatná odchylka

Určuje průměrnou odchylku od středu. Stačí odmocnit rozptyl. Bývá označována řeckým písmenem σ či s. Jedná se o často používanou míru variability, která nám vypovídá o tom, nakolik se od sebe navzájem liší jednotlivé prvky v souboru zkoumaných dat. V praxi lze říct je-li směrodatná odchylka malá, jsou si prvky souboru dat povětšinou navzájem podobné, naopak je-li velká jsou prvky odlišné.

$$\sigma = s \ = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{5.3}$$

Směrodatnou odchylku lze vypočíst pomocí funkce `numpy.std()` v `Pythonu`.

5.1.12 Variační koeficient

Slouží ke studiu, zda není s daty něco podivného. Vydělíme směrodatnou odchylku průměrem a počítáme jej v procentech. Hodnoty větší než 15 % až 30 % (záleží jakého charakteru mám data) svědčí o nějakém problému (třeba kdybychom porovnávali stáří babičky a vesmíru).

$$v_x = \frac{s_x}{\bar{x}} \times 100 \tag{5.4}$$

5.2 Distribuce pravděpodobnosti

Pro popis dat využíváme obvykle číselných hodnot, proto je pro popis některých dějů v přírodě nutno stanovit pravidla a každému jevu přiřadit hodnotu či pravděpodobnost s jakou ten jev nastává.

5.2.1 Náhodná veličina

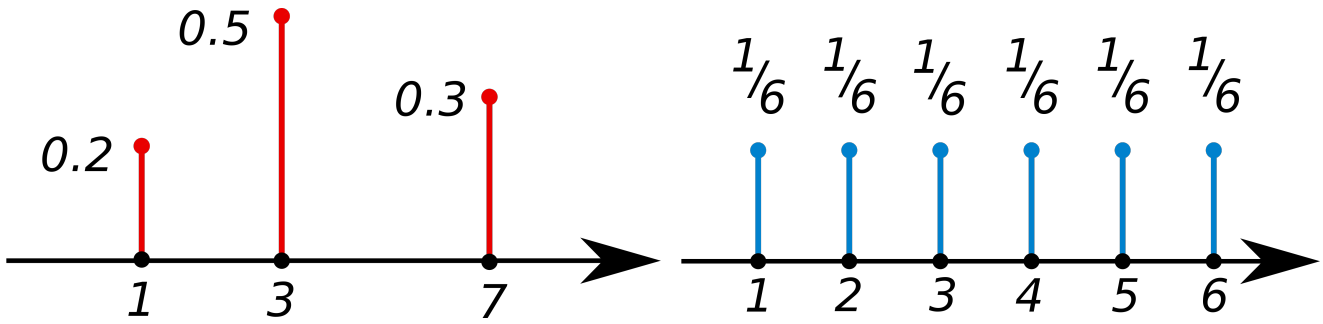
Náhodná veličina X je reálná funkce definovaná na množině všech elementárních jevů, která každému jevu přiřadí reálné číslo. Můžeme ji rozdělit na spojitou a diskrétní. V případě spojité náhodné veličiny je obor hodnot M otevřený nebo uzavřený interval. Pro diskrétní náhodnou veličinu je obor hodnot M konečná nebo nekonečná posloupnost.

Náhodnou veličinu lze také popsat jako libovolná veličinu, kterou je možné opakovaně měřit u různých objektů, v různých místech nebo v různém čase ale její hodnoty závisí na náhodě. Tyto výsledky lze pak následně zpracovat.

Příkladem může být házení mincí, kde jevu když padne rub přiřadíme číselnou hodnotu 0, a jevu, kdy padne líc přiřadíme 1.

5.2.2 Pravděpodobnostní funkce

Pravděpodobnostní funkce (anglicky *probability mass function*, zkráceně *PMF*) je funkce, která udává pravděpodobnost, že diskrétní náhodná veličina se přesně rovná nějaké hodnotě [1]. Pravděpodobnost jednotlivých hodnot x označujeme $P(x)$ a obvykle je zobrazujeme pomocí tabulky a grafu.



Obrázek 5.1: Pravděpodobnostní funkce (vlevo) a pravděpodobnostní funkce pro poctivou hrací kostku(vpravo).

5.2.3 Distribuční funkce diskrétní veličiny

Pomocí pravděpodobnostní funkce lze zavést distribuční funkci vztahem

$$F(x) = P[X \leq x] \tag{5.5}$$

Distribuční funkce je vždy neklesající, je spojitá zprava a její hodnoty leží v intervalu od 0 do 1.

Průběhy distribučních funkcí a jejich vlastnosti jsou dány typem jejich rozdělení, základní seznam diskrétních rozdělení je zde:

- Alternativní rozdělení
- Binomické rozdělení

- Poissonovo rozdělení
- Negativně binomické rozdělení
- Pascalovo rozdělení
- Geometrické rozdělení
- Hypergeometrické rozdělení
- Logaritmické rozdělení

Více informací o jednotlivých rozděleních naleznete v libovolné literatuře o teorii pravděpodobnosti či statistice.

5.2.4 Distribuční funkce spojité veličiny

Distribuční funkce anglicky označovaná jako *cumulative distribution function* zkráceně *CDF* je definována vztahem:

$$F_X(x) = P[X \leq x] \tag{5.6}$$

V případě, že X je spojitá náhodná proměnná s hustotou f, potom platí

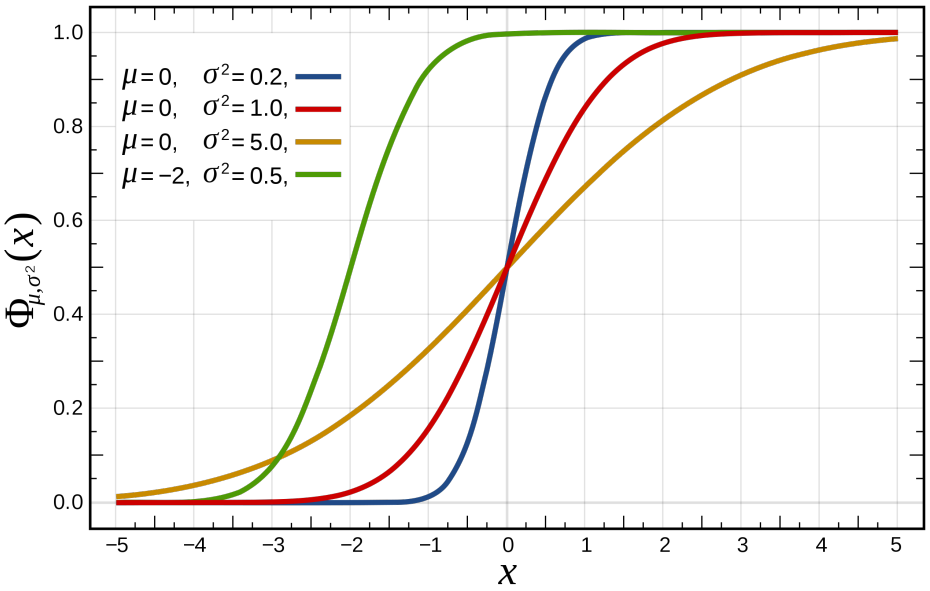
$$F_X(x) = \int_{-\infty}^{x} f(t)\mathrm{d}t \tag{5.7}$$

Průběhy distribučních funkcí a jejich vlastnosti jsou dány typem jejich rozdělení, základní seznam spojitých rozdělení je zde:

- Rovnoměrné rozdělení
- Normální (Gaussovo) rozdělení
- Logaritmicko-normální rozdělení
- Exponenciální rozdělení
- Cauchyho rozdělení
- Maxwellovo-Boltzmannovo rozdělení
- Studentovo rozdělení
- Fisherovo-Snedecorovo rozdělení
- χ^2 rozdělení (Chí kvadrát)

Více informací o jednotlivých rozděleních naleznete v libovolné literatuře o teorii pravděpodobnosti či statistice.

Pro vykreslení funkce za pomoci `Pythonu` využijeme knihovnu `SciPy` a její funkci `cdf(x, loc=0, scale=1)`.



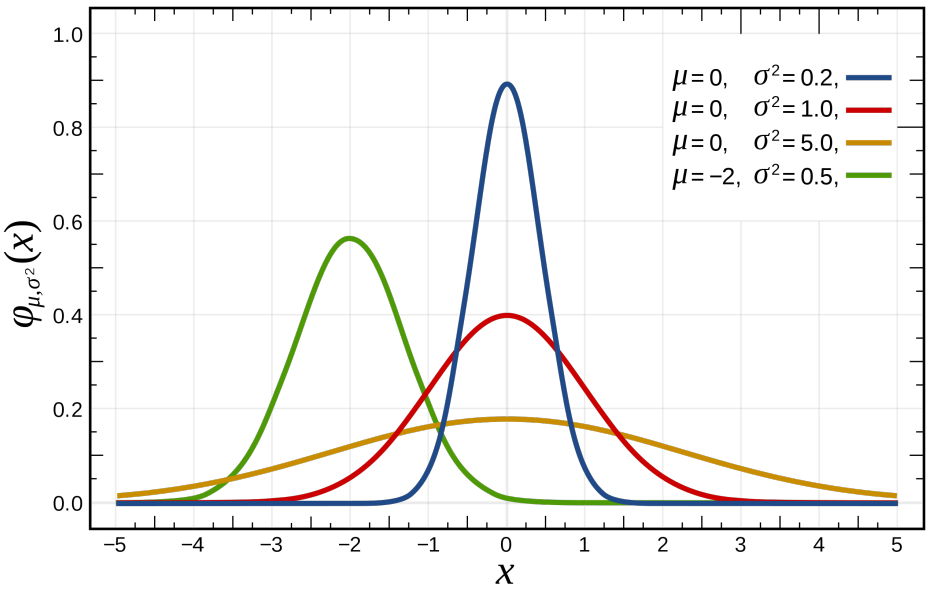
Obrázek 5.2: Příklady distribučních funkcí pro normální rozdělení.

5.2.5 Hustota pravděpodobnosti

Hustota pravděpodobnosti se anglicky nazývá *Probability Density Function* proto bývá označována jako *PDF*. Tuto funkci, když zintegrujeme na kterémkoli jejím vzorku, vyjde nám relativní pravděpodobnost, že hodnota náhodné proměnné by se rovnala tomuto vzorku.[2] Zobrazuje nám hustotu rozložení pravděpodobnosti.

$$\int_{\Omega} \rho(x)\mathrm{d}x = 1 \tag{5.8}$$

Pro vykreslení funkce za pomoci `Pythonu` využijeme knihovnu `SciPy` a její funkci `pdf(x, loc=0, scale=1)`.



Obrázek 5.3: Příklady hustot pravděpodobností pro normální rozdělení.

5.2.6 Centrální limitní věta

jsou-li X_i nezávislé náhodné veličiny s konečným rozptylem, pak výběrový průměr má při dostatečně velkém počtu pozorování přibližně normální rozdělení, ať už X_i pocházejí z libovolného rozdělení.

5.2.7 Normální rozdělení

Normální rozdělení neboli Gaussovo rozdělení je jedno z nejvýznamnějších rozdělení a to díky *Centrální limitní větě*. Toto rozdělení tak popisuje velké množství jevů a pro většinu aplikací si s ním vystačíme. Normální rozdělení je definováno 2 parametry (neboli charakteristikami) a to rozptylem σ^2 a střední hodnotouμ. Hustota pravděpodobnosti tohoto rozdělení má tvar

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{5.9}$$

Typickým příkladem normálního rozdělení je například výška dospělých lidí či velikost jejich inteligence. Jelikož nejvíce lidí má průměrnou výšku či inteligenci je křivka v tomto bodě nejvyšší a do obou stran postupně klesá směrem k nule. To vypovídá o tom, že extrémně malých či velkých lidí je velmi málo a to stejné platí i o inteligenci.

Úkoly

1. Budeme losovat z 5 bílých a 9 černých míčků. Náhodná veličina X nám představuje počet bílých míčků mezi 5 vybranými. Vytvořte pravděpodobnostní a distribuční funkci této náhodné veličiny.
2. Zkusme si následující pokus, kde budeme házet virtuální poctivou mincí. Hodíme třicetkrát mincí a budeme počítat kolikrát padne rub (1) či líc (0). Tento následně pokus budeme opakovat 100 krát. K tomu využijeme `Python` a vykreslete si hustotu pravděpodobnosti.

Souhrn kapitoly

Statistické zpracování dat je obsáhlá disciplína, jež by s klidem zabrala samostatný předmět, tato kapitola je pouze shrnutím základních a nejpoužívanějších pojmů používaných při základním zpracování dat.

Otázky

1. Jaké známe charakteristiky středu a jak se liší?
2. K čemu nám slouží směrodatná odchylka?
3. Proč je zrovna tak důležité Normální rozdělení?

Další informace

- PRESS William H., TEUKOLSKY Saul A., VETTERLING William T., FLANNERY Brian P. Numerical recipes. The art of scientific computing. Third edition. *Cambridge*. 2007, ISBN-13 978-0-521-88068-8

- HUFF, Darrell. Jak lhát se statistikou. *Nakladatelství BRÁNA A.S.*. Praha,2013. ISBN-978-80-7243-623-1
- FEYNMAN, Richard Phillips, Robert B. LEIGHTON a Matthew SANDS. Feynmanovy přednášky z fyziky: revidované vydání s řešenými příklady. 2. vydání. Přeložil Ivan ŠTOLL. Praha: *Fragment*, 2013. ISBN 978-80-253-1642-9.

Kapitola 6

Řešení okrajové úlohy relaxační metodou

Jiří Horák

Rychlý náhled: Tato kapitola se zabývá numerickým řešením systému obyčejných diferenciálních rovnic 1. řádu relaxační metodou, konkrétně řešení okrajové úlohy. Studenti se podrobně seznámí se způsobem implementace této metody v jazyce `Python`. Součástí je také vzorové řešení *Gorillas problému.*

Klíčová slova: numerické řešení diferenciálních rovnic, obyčejné diferenciální rovnice, relaxační metoda, okrajová úloha

Cíle studia: Po prostudování kapitoly budete schopní v `Pythonu` řešit systémy diferenciálních rovnic se zadanými okrajovými podmínkami, například pohyb těles v gravitačním poli země. Kapitola je dále základem pro následující kapitolu – Bondiho akrece, kde bude tato metoda využita na složitější systém rovnic.

6.1 Okrajová úloha

Relaxační metoda je užitečný nástroj pro řešení jedno-dimenzionálních okrajových úloh. Dále uvažujeme set obyčejných diferenciálních rovnic 1. řádu ve tvaru:

$$\frac{\mathrm{d}\mathbf{Y}}{\mathrm{d}x} = \mathbf{F}\left(x, \mathbf{Y}\right), \quad x_a \leq x \leq x_b, \tag{6.1}$$

kde x je nezávislá proměnná, $\mathbf{Y}(x) = [Y^0(x), \ldots, Y^{N-1}(x)]$ je vektor závislých proměných (proměnné jednotlivých řešení $Y^\alpha(x)$ jsou značeny řeckými indexy) a $\mathbf{F} : \mathbb{R}^{N+1} \to \mathbb{R}^N$ jsou nelineární funkce popisující pravé strany rovnic.

Set N diferenciálních rovnic by měl být ohraničen N okrajovými podmínkami (omezeními). Obecně mohou být okrajové podmínky z obou stran oblasti řešení (ve speciálním případě, kdy jsou všechny jen na jedné straně, se úloha stává počáteční úlohou). Obecně mohou být okrajové podmínky psány

$$\mathbf{A}\left[\mathbf{Y}\left(x_a\right)\right] = 0, \quad \mathbf{B}\left[\mathbf{Y}\left(x_b\right)\right] = 0, \tag{6.2}$$

kde $\mathbf{A}$ a $\mathbf{B}$ jsou dvě nelineární funkce, $\mathbf{A} : \mathbb{R}^N \to \mathbb{R}^{N_a}$ a $\mathbf{B} : \mathbb{R}^N \to \mathbb{R}^{N_b}$, kde N_a a N_b je počet omezení aplikován na x_a a x_b, a $N_a + N_b = N$.

6.1.1 Diskretizace

Soustavu obyčejných diferenciálních rovnic budeme řešit na nepravidelné síti tvořené M body $x_a = x_0 \dots x_i, \dots x_{M-1} = x_b$. Rovnice vyhodnotíme na středových bodech $(x_{i+1/2})$ každého segmentu. Derivace jsou jen přibližné, vypočítané pomocí metody centrální diference, hodnoty řešení $\mathbf{Y}_{i+1}$ jsou aritmetickým průměrem $\mathbf{Y}_i$ a $\mathbf{Y}_{i+1}$ na dvou sousedních bodech. To je

$$\left(\frac{\mathrm{d}\mathbf{Y}}{\mathrm{d}x}\right)_{i+1/2} \approx \frac{\mathbf{Y}_{i+1} - \mathbf{Y}_i}{x_{i+1} - x_i}, \quad \mathbf{F}_{i+1/2} \approx \mathbf{F}\left[\frac{1}{2}\left(x_i + x_{i+1}\right), \frac{1}{2}\left(\mathbf{Y}_{i+1} + \mathbf{Y}_i\right)\right]. \tag{6.3}$$

Takto obdržíme $N \times (M-1)$ algebraických rovnic popisujících obyčejné diferenciální rocnice v každém segmentu $(0 \le i \le M-2)$,

$$\mathbf{E}_i\left(\mathbf{Y}_i, \mathbf{Y}_{i+1}\right) \equiv \mathbf{Y}_{i+1} - \mathbf{Y}_i - \mathbf{F}_{i+1/2}\left(x_{i+1} - x_i\right) = 0, \tag{6.4}$$

plus v celkem N okrajových oblastech na koncích domény,

$$\mathbf{A}\left(\mathbf{Y}_0\right) = 0, \quad \mathbf{B}\left(\mathbf{Y}_{M-1}\right) = 0 \tag{6.5}$$

pro $N \times M$ hodnot všech nezávislých proměnných na všech bodech sítě Y_i^α. Formálně to můžeme zapsat:

$$\mathcal{G}(\mathcal{Y}) \equiv \begin{pmatrix} \mathbf{A}\left(\mathbf{Y}_0\right) \\ \mathbf{E}_0\left(\mathbf{Y}_0, \mathbf{Y}_1\right) \\ \vdots \\ \mathbf{E}_{M-2}\left(\mathbf{Y}_{M-2}, \mathbf{Y}_{M-1}\right) \\ \mathbf{B}\left(\mathbf{Y}_{M-1}\right) \end{pmatrix} = 0. \tag{6.6}$$

kde

$$\mathcal{Y} \equiv \left[\mathbf{Y}_0, \dots, \mathbf{Y}_{M-1}\right] = \left[Y_0^0, Y_0^1, \dots, Y_i^\alpha, \dots Y_{M-1}^{N-1}\right]. \tag{6.7}$$

6.1.2 Iterativní řešení

Rovnice $\mathcal{G}(\mathcal{Y}) = 0$ může být řešena iterativně použitím Newton-Ralphsonovy metody. Předpokládejme, že máme počáteční odhad řešení $\mathcal{Y}^{(n)}$. Hledáme vylepšené řešení $\mathcal{Y}^{(n+1)} = \mathcal{Y}^{(n)} + \delta\mathcal{Y}$ takové, že $\mathcal{G}(\mathcal{Y}^{(n+1)}) = 0$. Expanzí v lineárním řádu dostaneme

$$\mathcal{G}\left(\mathcal{Y}^{(n+1)}\right) = \mathcal{G}\left(\mathcal{Y}^{(n)}\right) + \sum_{i,\alpha} \frac{\partial \mathcal{G}}{\partial Y_i^\alpha} \delta Y_i^\alpha = 0. \tag{6.8}$$

To je lineární rovnice, která může být řešena použitím Gaussovy eliminace.

Všimněte si, že struktura matice $L \equiv \partial\mathcal{G}/\partial Y_i^\alpha$ je následující:

$$L = \begin{pmatrix} \frac{\partial A^\beta}{\partial Y^\alpha} & & & & \\ \frac{\partial E_0^\beta}{\partial Y_0^\alpha} & \frac{\partial E_0^\beta}{\partial Y_1^\alpha} & & & \\ & \frac{\partial E_1^\beta}{\partial Y_1^\alpha} & \frac{\partial E_1^\beta}{\partial Y_2^\alpha} & & \\ & & \cdots & \cdots & \\ & & & \frac{\partial E_{M-2}^\beta}{\partial Y_{M-2}^\alpha} & \\ \frac{\partial E_{M-2}^\beta}{\partial Y_{M-1}^\alpha} & & & & \\ \frac{\partial B^\beta}{\partial Y^\alpha} & & & & \end{pmatrix}. \tag{6.9}$$

Je tomu tak proto, že $\mathbf{E}_i$ závisí pouze na hodnotách řešení ve dvou sousedních bodech sítě. Máme

$$\frac{\partial E_i^\beta}{\partial Y_j^\alpha} = \delta_\alpha^\beta \left(\delta_{i+1}^j - \delta_i^j\right) - \frac{1}{2}\left(\frac{\partial F^\beta}{\partial Y^\alpha}\right)_{i+1/2} (x_i + x_{i+1}) \left(\delta_i^j + \delta_{i+1}^j\right) \tag{6.10}$$

6.2 Implementace

Nejprve importujeme nezbytné moduly `numpy` pro manipulaci s vektory a maticemi, a `matplotlib` pro tvorbu grafů.

```
1 import numpy as np
2 import math
3 import matplotlib
4 import matplotlib.pyplot as plt
5
6 # some settings of matplotlib:
7 from matplotlib import rc
8 rc('text', usetex=True)       # ... in order to use LaTex fonts in the graphs.
9 rc('legend', frameon=False)   # ... as a default, legends without the boundary frame.
```

Nyní implementujeme jeden krok relaxace. Funkce `relaxation_step` bere jako vstup:

- Array hodnot nezávislé proměnné, $X = [x_0, \ldots, x_{M-1}]$.
- Vektor řešení všech hodnot závislé proměnné, $\mathcal{Y} = [Y_0^0, Y_0^1, \ldots, Y_i^\alpha, \ldots Y_{M-1}^{N-1}]$ popisující počáteční odhad.
- Pravou stranu rovnice $\mathbf{F}(x, \mathbf{Y}) = [F^1(x, \mathbf{Y}), \ldots F^{N-1}(x, \mathbf{Y})]$.
- Jakobián pravé strany rovnice jacF $= \partial(F^0, \ldots, F^{N-1})/\partial(Y^0, \ldots, Y^{N-1})$
- Dvě funkce $\mathbf{A}$ a $\mathbf{B}$, udávající okrajové podmínky na obou stranách oblasti řešení, $\mathbf{A}(\mathbf{Y}) = 0$ v $x = x_a$ a $\mathbf{B}(\mathbf{Y}) = 0$ v $x = x_b$, a jejich jakobiány $\partial(A^0, \ldots, A^{N_a - 1})/\partial(Y^0, \ldots, Y^{N-1})$ a $\partial(B^0, \ldots, B^{N_b - 1})/\partial(Y^0, \ldots, Y^{N-1})$.

Funkce vyhodnotí vektor $\mathcal{G}$ a jakobián L. Nakonec řeší rovnici

$$L\delta\mathcal{Y} = -\mathcal{G} \tag{6.11}$$

Výstup funkce je vektor $\delta\mathcal{Y}$.

```
def relaxation_step(X, Y, F, jacF, A, jacA, B, jacB):

    # We find the number of mesh points M and number of the solution variables N
    # from the sizes of the solution vectors X and Y:
    M = len(X)
    N = len(Y)//M
    if len(Y)%M != 0:
        raise ValueError('Incompatible sizes of X and Y arrays.')

    # Create the empty vector G and empty matrix L:
    G = np.zeros(N*M)
    L = np.zeros((N*M, N*M))

    # Filling G and L:
    #------------------

    # We start with the constrains at left border of the domain (the block dA/dY):
    # We assign the number of constraints Na as a size of A(Y_0)

    # Evaluating A(Y_0).
    # Note that vector Y_0 corresponds to first N elements of the vector Y.
    Aval = A(Y[0:N])
    Na = len(Aval)
    if Na > 0:
        G[0:Na] = Aval
        L[0:Na, 0:N] = jacA(Y[0:N])

    # Now we calculate and place contributions of the individual segments:
    # (the index i numbers the segments)
    for i in range(M-1):
        # Find the midpoint (x) and the size of the segment (dx):
        x = (X[i+1] + X[i])/2
        dx = X[i+1] - X[i]
        # Find the average value of the solutions y = (Y_i + Y_{i+1})/2
        # and the difference dy = Y_{i+1} - Y_i.
        # Note that the vector Y_i is located in the solution vector Y[] at
        # indices i*N, i*N+1, ..., i*N+N-1.
        y = (Y[i*N:(i+1)*N] + Y[(i+1)*N:(i+2)*N])/2
        dy = Y[(i+1)*N:(i+2)*N] - Y[i*N:(i+1)*N]
        # Fill the vector G: Note that the contributions has to be placed AFTER the
        # boundary conditions (first Na elements)
        G[Na+i*N:Na+(i+1)*N] = dy - F(x, y)*dx
        # Fill the matrix L:
        # First calculate the gradient of F with respect to the solution variables,
        jF = jacF(x, y)
        delta = np.identity(N)
        L[Na+i*N:Na+(i+1)*N, i*N:(i+1)*N]     = -dx*jF/2 - delta
```

```
48          L[Na+i*N:Na+(i+1)*N, (i+1)*N:(i+2)*N] = -dx*jF/2 + delta
49
50      # Finally the constrains at right end of the domain:
51      if Na < N:
52          # There should be N-Na constraints.
53          # Evaluate the boundary function B(Y_{M-1}) & Jacobian:
54          Bval = B(Y[N*(M-1):])
55          Nb = len(Bval)
56          if Na + Nb != N:
57              raise
                ↪ ValueError('The number of boundary constraints ({}) does not match the number of variables
                ↪ N))
58          G[Na+(M-1)*N:] = Bval
59          L[Na+(M-1)*N:,N*(M-1):] = jacB(Y[N*(M-1):])
60
61      # The matrix L and the vector G is now assembled.
62      # Solve L.dY = -G for the improvement dY:
63      dY = np.linalg.solve(L, -G)
64
65      return dY
```

Tato funkce je implementována v modulu `relaxation.py`.

Gorilla problem

Kdysi dávno existovala velmi populární počítačová hra napsaná v `MS-DOS q-basic`, jmenovala se `Gorillas` nebo `GORILLAS.BAS`, podle jména souboru se zdrojovým kódem. Hra byla velmi jednoduchá. Byli v ní dvě gorily, které po sobě házely vybuchující banány nad obrysem města. Hra byla určená pro dva hráče, každý musel zadat počáteční rychlost a úhel hodu tak, aby trefil druhého hráče.

Pro ty z vás, kteří se zajímají o počítačovou prehistorii, tady je odkaz na online verzi: https://www.retrogames.cz/play$_6$54 – *DOS.php*

Chceme řešit stejný problém použitím relaxační metody. Podívejme se na problém takto: Vezmeme střelce (první gorilu) a cíl (druhou gorilu) ve vzdálenosti d. Pro jednoduchost, předpokládejme že střelec i cíl jsou ve stejné výšce a zanedbejme jakoukoli interakci objektů (banánů) s okolním vzduchem. Jaká má být počáteční rychlost a úhel tak, abychom trefili cíl? Protože problém vede na nekonečný počet řešení, musíme specifikovat čas letu objektu t_b.

Ze znalostí základní fyziky víme, že pohyb bude určen tímto systémem diferenciálních rovnic:

$$\ddot{x} = 0, \quad \ddot{y} = -g, \tag{6.12}$$

kde $x(t)$ a $y(t)$ popisují trajektorii banánu a g je tíhové zrychlení. Naše situace je popsána čtyřmi okrajovými podmínkami

$$x(0) = 0, \quad y(0) = 0, \quad x(t_b) = d, \quad y(t_b) = 0. \tag{6.13}$$

Dvě z nich jsou určeny v $t = 0$, delší dvě v $t = T$. Tyto představují okrajové podmínky úlohy.

Chceme úlohy řešit s použitím relaxační metody. Proto musíme transformovat systém dvou obyčejných diferenciálních rovnic 2. řádu na systém čtyř obyčejných diferenciálních rovnic 1. řádu. Nejjednodušší

způsob jak to provést, je zavést rychlosti $u = \dot{x}$ a $v = \dot{y}$ jako nové závislé proměnné. Pak dostaneme

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} x \\ u \\ y \\ v \end{pmatrix} = \begin{pmatrix} u \\ 0 \\ v \\ -g \end{pmatrix}. \tag{6.14}$$

Dále zavedeme vzorkování časového intervalu $[0, t_b]$. Použijeme rovnoměrně rozloženou síť bodů, takže vektor hodnot nezávislé proměnné je $T = [0, \ldots, t_i, \ldots, t_{M-1} = t_b]$. Vektor lokálního řešení je jednoduše $\mathbf{Y} = [x, u, y, v]^T$. Vektor celkového řešení má tvar

$$\mathcal{Y} = [x_0, u_0, y_0, v_0, \ldots, x_i, u_i, y_i, v_i, \ldots x_{M-1}, u_{M-1}, y_{M-1}, v_{M-1}].$$

Funkce $\mathbf{F}$, $\mathbf{A}$ a $\mathbf{B}$ popisující pravé strany rovnic a okrajové podmínky jsou

$$\mathbf{F}(t, [x, u, y, v]) = [u, 0, v, -g], \quad \mathbf{A}([x_0, u_0, y_0, v_0]) = [x_0, y_0] = 0,$$

$$\mathbf{B}([x_{M-1}, u_{M-1}, y_{M-1}, v_{M-1}]) = [x_{M-1}, y_{M-1}] = 0,$$

a odpovídající Jakobiány:

$$\frac{\partial \mathbf{J}}{\partial \mathbf{Y}} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad \frac{\partial \mathbf{A}}{\partial \mathbf{Y}} = \frac{\partial \mathbf{B}}{\partial \mathbf{Y}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Pro použití relaxační metody musíme poskytnout počáteční odhad, který potenciálně povede ke skutečnému řešení. Počáteční odhad není pravým řešením diferenciálních rovnic, ale obvykle stačí, když splňuje základní topologické vlastnosti předpokládaného řešení (například, že spojitě spojuje počáteční a koncové okrajové podmínky). My použijeme obzvláště jednoduchý první odhad – přímku. Pak předpokládáme, že se banán hýbe s konstantní rychlostí horizontálním směrem a dosáhne cíle v čase t_b. Tato situace odpovídá stavu bez gravitačního pole. Horizontální rychlost je v tomto případě dána jako $u = d/t_b$, vertikální rychlost je nulová a trajektorie je popsána $x(t) = td/t_b$ a $y = 0$. Řešení je implementováno dále.

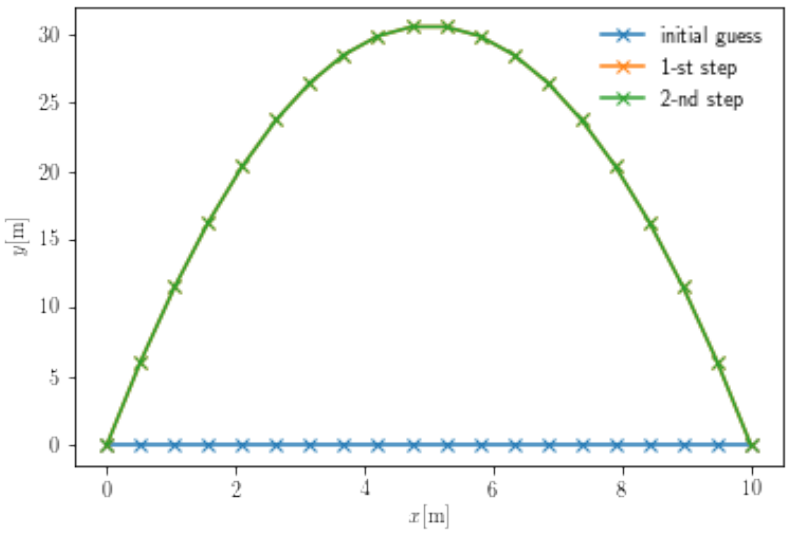
```
1  class GorillasProblem:
2
3      def __init__(self, d, g, tb, M):
4          """
5          Initialization.
6          `d`  is the distance of the target,
7          `g`  is the gravitational acceleration
8          `tb` is the time, when the should reach the target,
9          `M`  is the number of mesh points in time domain.
10         """
11         self._d = d
12         self._g = g
13         # create evenly spaces sample for the independent variable (time):
14         self._T = np.linspace(0, tb, M)
15         # put as the solution vector the initial guess:
16         self._Y = self.initial_guess()
```

```
17          self._norm_dY = None
18
19      def initial_guess(self):
20          """  Return the global solution vector Y corresponding
21               to the initial guess. """
22          u0 = self._d/self._T[-1]           # ...constant horizontal speed
23          Y = [0, u0, 0, 0]
24          for t in self._T[1:]:
25              Y += [u0*t, u0, 0, 0]
26          return np.array(Y)
27
28      def F(self, x, Y):
29          """ The right-hand sides of the differential equations. """
30          [x, u, y, v] = Y
31          return np.array([u, 0, v, -self._g])
32
33      def jacF(self, x, Y):
34          """ The Jacobian of the right-hand side. """
35          return np.array([[0, 1, 0, 0], [0, 0, 0, 0], [0, 0, 0, 1], [0, 0, 0, 0]])
36
37      def A(self, Y):
38          """ The boundary constrains at t = 0: x=0 and y=0 """
39          [x, u, y, v] = Y
40          # We return vector of just 2 items as there are only
41          # two constraints: x = 0 and y = 0.
42          return np.array([x, y])
43
44      def jacA(self, Y):
45          """ Jacobian of the boundary constrains at t=0. """
46          return np.array([[1, 0, 0, 0], [0, 0, 1, 0]])
47
48      def B(self, Y):
49          """ The boundary constrains at t = tb: z=d and y=0 """
50          [x, u, y, v] = Y
51          return np.array([x - self._d, y])
52
53      def execute_one_step(self):
54          """ Executes the relaxation method. """
55          dY = relaxation_step(self._T, self._Y, self.F, self.jacF, self.A, self.jacA, self.B, self.jacA)
56          self._norm_dY = np.linalg.norm(dY)
57          self._Y += dY
58
59      def norm_dY(self):
60          """ Returns the L2-norm of the last update. """
61          return self._norm_dY
62
63      def initial_velocity(self):
64          """ Returns the initial velocity of the current solution and the angle. """
65          u = self._Y[1]
66          v = self._Y[3]
67          return math.hypot(u, v), math.atan2(v, u)*180/math.pi
68
69      def plot_solution_xy(self, ax, **kwargs):
```

```
70          """ Plots the solution into the axes object `ax` """
71          ax.plot(self._Y[0::4], self._Y[2::4], **kwargs)
```

A jeden příklad pro parametry $d = 10\,\mathrm{m}$, $g = 9.8\,\mathrm{m}\cdot\mathrm{s}^{-2}$ a $t_b = 5\,\mathrm{s}$.

```
1  gorillas = GorillasProblem(10, 9.8, 5, 20)
2  fig, ax = plt.subplots(1)
3  gorillas.plot_solution_xy(ax, ls='-', marker='x', label='initial guess')
4
5  gorillas.execute_one_step()
6  gorillas.plot_solution_xy(ax, ls='-', marker='x', label='1-st step')
7  print('1-st step norm = {}'.format(gorillas.norm_dY()))
8
9  gorillas.execute_one_step()
10 gorillas.plot_solution_xy(ax, ls='-', marker='x', label='2-nd step')
11 print('2-nd step norm = {}'.format(gorillas.norm_dY()))
12
13 ax.set_xlabel(r'$x [\mathrm{m}]$')
14 ax.set_ylabel(r'$y [\mathrm{m}]$')
15 ax.legend()
16 print('initial velocity = {} m/s, angle = {} deg'.format(*gorillas.initial_velocity()))
```



V tomto případě konvergovala relaxační metoda hned po prvním kroku. Je to díky tomu, že systém obyčejných diferenciálních rovnic je homogenní a lineární. Newton-Raphsonova metoda aproximující funkci $\mathcal{G}(\mathcal{Y})$ tak dává přímo přesné řešení. To však není obvyklý případ, jak uvidíte při řešení následujícího problému.

6.3 Numerické nebo automatické derivace

Vypočítat Jakobián je jednoduché, avšak někdy to může být zdlouhavé. Místo přímého zapsání vzorce můžeme použít numerickou diferenciaci:

Zavedeme numerickou derivaci funkce jedné proměnné (`numder`) a gradient $\partial_k F$ pro funkci více proměnných. F může být také vektor nebo tenzor.

```
1  def numder(f, x, dx=1e-10):
2      return (f(x+dx) - f(x-dx))/(2*dx)
3
4  def grad(F, X, eps=1e-5):
5      n = len(X)
6      result = []
7      for i in range(n):
8          FF = lambda t: F(np.concatenate((X[:i], [t], X[i+1:]), axis=None))
9          result.append(numder(FF, X[i], dx=eps))
10     return np.array(result)
```

Úkol

Zkuste znova vyřešit Gorilla problém, avšak bez zanedbání odporu vzduchu. Navíc uvažujte nenulovou rychlost větru a různou výšku střelce a terče.

Souhrn kapitoly

V této kapitole jsme se seznámili s metodou řešení soustavy obyčejných diferenciálních rovnic pomocí metody relaxace, také jsme si zopakovali použití Newton-Raphsonovy metody. Dále poznatky využijeme v následující kapitole.

Otázky

- Kolik okrajových podmínek musíme znát, abychom mohli řešit systém N obyčejných diferenciálních rovnic 1. řádu?
- Jak se nazývá úloha, která má všechny okrajové podmínky na jedné straně oblasti řešení?
- Jakým způsobem je možné vyřešit problém, který je zadán soustavou obyčejných diferenciálních rovnic vyššího řádu?

Kapitola 7

Bondiho akrece

Jiří Horák

Rychlý náhled: V této kapitole budeme vytvářet numerický model *Bondiho akrece* v jazyce `Python`. Využijeme k tomu relaxační metodu popsanou v předcházející kapitole. Řešení budeme implementovat pro síť souřadnic s logaritmickým rozložením.

Klíčová slova: Bondiho akrece, numerické modelování, numerické řešení diferenciálních rovnic, relaxační metoda.

Cíle studia: V této kapitole najdete konkrétní implementaci a použití relaxační metody na řešení reálného fyzikálního problému, konkrétně Bondiho akrece. Také se naučíte definovat a používat různé rozložení sítě souřadnic.

7.1 Akreční procesy

Akrece je proces, při němž se hmota akumuluje na povrchu hmotného tělesa díky gravitační přitažlivé síle. K akreci dochází nejčastěji ve vesmíru v okolí velmi hmotných těles, jako jsou hvězdy, případně i hmotné planety nebo extrémní objekty (černé díry, neutronové hvězdy). V případě hvězd nebo planet dochází k akreci prachových částic, v případě kompaktních objektů se jedná o plasma, díky extrémním tlakům a teplotám, které v jejich okolí nalezneme.

Specifický případ, který je zároveň nejjednodušší, je případ stacionární sféricky symetrické (Bondiho) akrece.

7.2 Sférická akrece

Uvažujeme nerelativistickou, sféricky symetrickou akreci polytropické kapaliny na kompaktní objekt.

Definice

Polytropická kapalina je popsána polytropickou stavovou rovnicí

$$p = K\rho^{\gamma}, \tag{7.1}$$

kde ρ a p je hustota a tlak plynu, K je polytropická konstanta a γ je polytropický index.

Polytropická kapalina je jednoduchý model kapaliny, pro který různé hodnoty polytropického indexu odpovídají různým druhům kapalin. Rychlost a, kterou se v kapalině šíří zvuk, je dána

$$a^2 = \frac{\mathrm{d}p}{\mathrm{d}\rho} = \gamma K \rho^{\gamma-1} = \frac{\gamma p}{\rho}. \tag{7.2}$$

Definice

Gravitační pole kompaktního objektu je popsáno Newtonovským gravitačním potenciálem

$$\Phi(r) = -\frac{GM}{r}, \tag{7.3}$$

kde M je hmotnost kompaktního objektu, G je gravitační konstanta a r je radiální souřadnice.

Předpokládáme, že ve velké vzdálenosti od centra je tok v klidu, takže radiální rychlost u^r vymizí. Asymptotické hodnoty hustoty a tlaku označíme ρ_∞ a p_∞.

Dynamika kapaliny je dána rovnicí kontinuity popisující zachování hmoty, a radiální Eulerovou rovnicí popisující zachování hybnosti v radiálním směru. Rovnice kontinuity má tvar

$$\vec{\nabla} \cdot (\rho \vec{v}) = \frac{1}{r^2} \frac{\mathrm{d}}{\mathrm{d}r} \left(r^2 \rho u^r \right) = 0 \tag{7.4}$$

V tomto případě může být dále integrována

$$r^2 \rho u^r = \text{const} \equiv -\frac{\dot{M}}{4\pi}. \tag{7.5}$$

Význam konstanty $\dot{M}$ je jasný: udává, jak rychle je hmota dodávána centrálnímu objektu. Rovnice (7.5) udává pouze celkové množství hmoty, které překročí sféru o poloměru r za jednotku času, $\dot{M} = 4\pi r^2 \rho(-u^r)$ je konstantní bez ohledu na velikost sféry. Tato podmínka musí být splněna, jinak by došlo k akumulaci hmoty v některých místech toku a proces by nebyl stacionární.

Radiální Eulerova rovnice popisuje pohyb elementu kapaliny pod vlivem tlakové a gravitační síly

$$u^r \frac{\mathrm{d}u^r}{\mathrm{d}r} = -\frac{1}{\rho}\frac{\mathrm{d}p}{\mathrm{d}r} - \frac{GM}{r^2} = -a^2 \frac{\mathrm{d}\ln\rho}{\mathrm{d}r} - \frac{GM}{r^2}. \tag{7.6}$$

Použijeme–li logaritmus radiální derivace rovnice kontinuity, dostaneme

$$\frac{\mathrm{d}\ln\rho}{\mathrm{d}r} = -\frac{2}{r} - \frac{1}{u^r}\frac{\mathrm{d}u^r}{\mathrm{d}r} \tag{7.7}$$

a radiální Eulerova rovnice má pak tvar

$$\frac{\mathrm{d}(u^r)^2}{\mathrm{d}r} = \frac{\frac{4a^2}{r} - \frac{2GM}{r^2}}{1 - \frac{a^2}{(u^r)^2}}. \tag{7.8}$$

7.2.1 Bezrozměrná verze

Definujeme-li následující bezrozměrné proměnné:

$$y \equiv \left(\frac{-u^r}{a_\infty}\right)^2, \quad \tilde{a} \equiv \frac{a}{a_\infty}, \quad x \equiv \frac{r}{r_*}, \quad \dot{m} \equiv \frac{\dot{M}}{4\pi r_*^2 \rho_\infty a_\infty}, \tag{7.9}$$

kde

$$r_* \equiv \frac{GM}{a_\infty^2} \tag{7.10}$$

je přirozená délková, je možné celou situaci popsat jednou obyčejnou diferenciální rovnicí (Bondiho rovnice) a jednou algebraickou rovnicí

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{\frac{4\tilde{a}^2}{x} - \frac{2}{x^2}}{1 - \frac{\tilde{a}^2}{y}}, \quad \tilde{a}^2 = \left(\frac{\dot{m}}{x^2 y^{1/2}}\right)^{\gamma-1}. \tag{7.11}$$

Okrajové podmínky při $x \to \infty$ vyplývají z požadavku nulové radiální rychlosti ve velké vzdálenosti, $(y \to 0)$, rychlost zvuku pak nabývá asymptotické hodnoty $\tilde{a} \to 1$. Druhá podmínka dává asymptotické chování

$$y \approx \left(\frac{\dot{m}}{x^2}\right)^2 \quad (x \to \infty). \tag{7.12}$$

7.2.2 Kritický bod a podmínka regularity

Bezrozměrná Bondiho rovnice má kritický bod, ve kterém rychlost proudění nabývá lokální rychlosti zvuku, $y = \tilde{a}^2$. V tom případě je jmenovatel na pravé straně rovnice (7.11) nulový a řešení rovnice je může být regulární pouze pokud je ve stejném bodě nulový také čitatel. Takové místo nazýváme jako sonický bod a příslušnou radiální souřadnici označíme x_s. Aby bylo řešení regulární, musí tedy platit

$$\frac{4\tilde{a}^2}{x_\mathrm{s}} - \frac{2}{x_\mathrm{s}^2} = 0 \quad \Leftrightarrow \quad y = \tilde{a}^2. \tag{7.13}$$

Z toho vyplývá další podmínka na řešení Bondiho rovnice. Obecně platí, že integrační křivka $y(x)$, která prochází spojitě sonickým bodem (a splňuje podmínku regularity určenou výše), nemusí být v souladu s požadovaným asymptotickým chováním řešením při $x \to \infty$. Obě tyto podmínky jsou tak splněny jen při určitých hodnotách akreční rychlosti $\dot{m}$, která hraje roli vlastní hodnoty problému. Stacionární akrece je možná jen pro tuto hodnotu akrečního toku, při jiných hodnotách nemusí být akrece nutně stacionární.

7.3 Analytické řešení

Další podmínka regularity představuje zajímavou výzvu při numerickém řešení. Dříve než se o něj pokusíme, nastíníme si analytické řešení, abychom měli něco k porovnání. Hlavní výhoda sférické akrece je, že je možné integrovat také rovnici pro hybnost, takže se jedná o algebraický problém (přestože je dán transcendentní rovnicí).

Začněme s bezrozměrnou rovnicí pro hybnost

$$y^{1/2}\frac{\mathrm{d}y^{1/2}}{\mathrm{d}x} + \frac{1}{\rho a_\infty^2}\frac{\mathrm{d}p}{\mathrm{d}x} + \frac{1}{x^2} = 0. \tag{7.14}$$

Je snadné integrovat první a třetí člen podle x. Druhý člen lze integrovat následovně:

$$\frac{1}{a_\infty^2}\int \frac{1}{\rho}\frac{\mathrm{d}p}{\mathrm{d}x}\mathrm{d}x = \frac{1}{a_\infty^2}\int \frac{\mathrm{d}p}{\rho} = \frac{\gamma K}{a_\infty^2}\int \rho^{\gamma-2}\mathrm{d}\rho = \frac{\gamma K \rho^{\gamma-1}}{(\gamma-1)a_\infty^2} = \frac{\tilde{a}^2}{\gamma-1}. \tag{7.15}$$

Pak dostaneme

$$\frac{y}{2} + \frac{\tilde{a}^2}{\gamma-1} - \frac{1}{x} = \mathrm{const} \equiv \Psi, \tag{7.16}$$

což je bezrozměrná Bernoulliho rovnice. Integrační konstanta Ψ označuje odlišné integrační křivky, které jsou řešením $y(x)$ Bernoulliho rovnice (7.11). Ta z nich, která odpovídá naší okrajové podmínce, musí splňovat $y \to 0$ a $\tilde{a} \to 1$ pro $x \to \infty$, je tedy dána hodnotou

$$\Psi \equiv \Psi_0 = 1/(\gamma - 1). \tag{7.17}$$

Které hodnotě Ψ odpovídají řešení, které spojitě prochází kritickým bodem? Z podmínky regularity víme, že

$$y_\mathrm{s} = \tilde{a}_\mathrm{s}^2, \quad x_\mathrm{s} = \frac{1}{2\tilde{a}_\mathrm{s}^2}. \tag{7.18}$$

Rovnice pro zachování hmoty nám dále dovoluje vyjádřit rychlost zvuku v sonickém bodě pomocí akreční rychlosti jako

$$\tilde{a}_\mathrm{s} = (4\dot{m})^{\frac{\gamma-1}{5-3\gamma}}. \tag{7.19}$$

Po dosazení do Bernoulliho rovnice dostáváme

$$\Psi \equiv \Psi_\mathrm{s} = \frac{\tilde{a}_\mathrm{s}^2}{2} + \frac{\tilde{a}_\mathrm{s}^2}{\gamma - 1} - 2\tilde{a}_\mathrm{s}^2 = \frac{5-3\gamma}{2\gamma-2}\tilde{a}_\mathrm{s}^2 = \frac{5-3\gamma}{2\gamma-2}(4\dot{m})^{\frac{2\gamma-2}{5-3\gamma}}. \tag{7.20}$$

Jelikož požadujeme, aby řešení procházelo spojitě sonickým bodem, a zároveň dávalo správné chování na velkých poloměrech, musí platit $\Psi_\mathrm{s} = \Psi_0$. To nám dává jednoduché analytické vyjádření polohy sonického bodu x_s a lokální rychlost zvuku $\tilde{a}_\mathrm{s}$ v sonickém bodě,

$$x_\mathrm{s} = \frac{5-3\gamma}{4}, \quad \tilde{a}_\mathrm{s}^2 = \frac{2}{5-3\gamma}. \tag{7.21}$$

Tato rychlost zvuku odpovídá akreční rychlosti $\dot{m}_0$

$$\dot{m}_0 = \frac{1}{4}\left(\frac{2}{5-3\gamma}\right)^{\frac{5-3\gamma}{2\gamma-2}}, \tag{7.22}$$

což je vlastní hodnota problému.

7.3.1 Asymptotické chování řešení:

Z předchozích odstavců už víme, jak se má řešení chovat ve velkých vzdálenostech od kompaktního objektu (když $x \to \infty$). Jaké chování lze očekávat, když $x \to 0$? K nalezení odpovědi se skvěle hodí Frobeniova teorie. Předpokládejme, že řešení blízko bodu $x = 0$ vystihuje závislost $y \approx A/x^\alpha$. Z Bernoulliho rovnice vybereme nejrychleji rostoucí členy, které musí být v rovnováze a jejich porovnáním určíme α a A. Z rovnice kontinuity je vidět, že $\tilde{a}^2 \sim x^{(\alpha-2)(\gamma-1)}$. První člen Bernoulliho rovnice (člen odpovídající kinetické energii) roste jako $x^{-\alpha}$, druhý člen (odpovídající vnitřní energii) roste jako $x^{(\alpha-2)(\gamma-1)}$ a poslední třetí člen (odpovídající potenciální energii) roste jako x^{-1}. Porovnáním exponentů za předpokladu že $1 \leq \gamma \leq 5/3$ zjistíme, že nejvyšší hodnota α odpovídá rovnováze mezi kinetickým a potencionálním členem a odpovídá $\alpha = 1$ a $A = 2$, takže

$$y \approx \frac{2}{x} \quad (x \to 0). \tag{7.23}$$

Odsud vidíme, že velmi blízko kompaktního objektu má největší vliv gravitace a tlak způsobuje jen zanedbatelné perturbace, zatímco ve velkých vzdálenostech je nejvýraznější rovnováha mezi gravitační potenciální a vnitřní energií. V obou případech má navíc zanedbávaný člen tendenci snižovat hodnoty rychlosti. To se bude hodit při numerickém řešení Bernoulliho rovnice pomocí metody půlení intervalů.

7.3.2 Grafické znázornění analytických výsledků

Na obrázku 7.1 je znázorněno analytické řešení popsáno dříve. Abychom graf mohli vytvořit, začneme nahráním potřebných modulů:

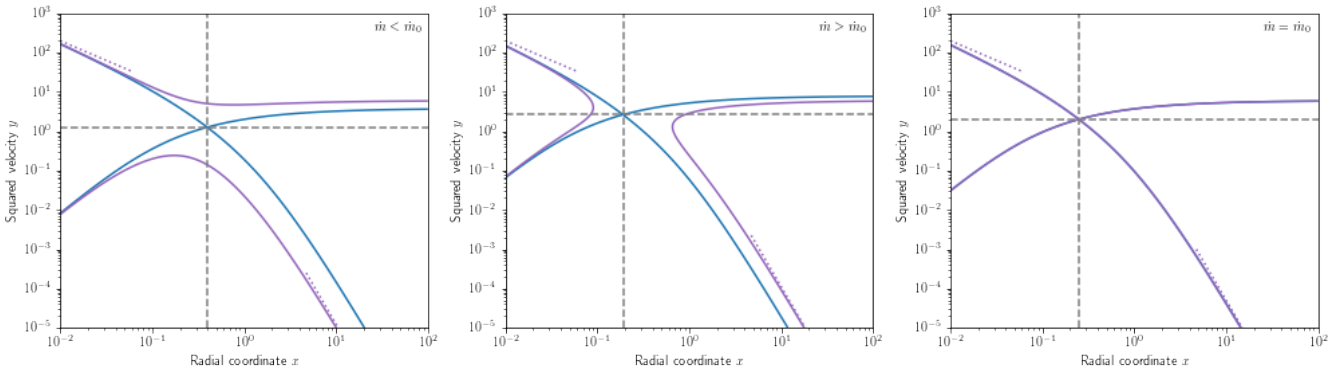
```
import numpy as np
import math
import scipy.optimize as opt
import matplotlib
import matplotlib.pyplot as plt

# in order to use LaTex fonts in the graphs:
from matplotlib import rc
rc('text', usetex=True)
rc('legend', frameon=False)

# introduce "nice" colors:
purple = '#9467bd'
blue = '#1f77b4'
```

V `Pythonu` dále definujeme funkce pro různé fyzikální veličiny popisující akreční tok. V obecném případě je akreční tok popsán dvěma parametry – polytropickým indexem γ a bezrozměrnou akreční rychlostí $\dot{m}$. Regulární řešení však existuje jen pro jednu konkrétní hodnotu $\dot{m}_0$, která je implementována jako funkce `mdot0`. Bernoulliho konstanta Ψ, jako funkce bezrozměrného poloměru x a kvadratické rychlosti y, je implementována pomocí funkce `Psi`. Její dvě speciální hodnoty Ψ_s a Ψ_0, odpovídající integrální křivce procházející sonickým bodem a řešení se správným chováním na velkých vzdálenostech, jsou dány funkcemi `Psis` a `Psi0`. Asymptotická chování řešení blízko kompaktního objektu a v nekonečnu popisují funkce `yasy_0` a `yasy_inf`, poloha kritického bodu $[x_\mathrm{s}(\gamma,\dot{m}), y_\mathrm{s}(\gamma,\dot{m})]$ je dána funkcemi `xs` a `ys`.

```
def mdot0(gamma):
    return (2/(5-3*gamma))**((5-3*gamma)/(2*gamma-2))/4.
def Psi(x, y, gamma, mdot):
    return y/2. - 1./x + (mdot/(x*x*math.sqrt(y)))**(gamma-1)/(gamma-1)
def Psis(gamma, mdot):
    return (5-3*gamma)/(2*gamma-2)*(4.*mdot)**((2*gamma-2)/(5-3*gamma))
def Psi0(gamma):
    return 1/(gamma-1)
def yasy_inf(x, mdot):
    return (mdot/(x*x))**2
def yasy_0(x):
    return 2/x
def xs(gamma, mdot):
    return 1/(2*ys(gamma, mdot))
def ys(gamma, mdot):
    return (4*mdot)**((2*gamma-2)/(5-3*gamma))
```

Nyní vykreslíme grafické řešení pro různé hodnoty akrečního toku. Místo řešení Bernoulliho rovnice, abychom získali specifické funkce $y = y(x;\Psi)$, vykreslíme čáry konstantního Ψ v rovině (x, y). Ukážeme si tři případy. První dva jsou nefyzikální, s $\dot{m} < \dot{m}_0$ odpovídající $\Psi_0 < \Psi_\mathrm{s}$ a $\dot{m} > \dot{m}_0$, což odpovídá $\Psi_0 < \Psi_\mathrm{s}$. Nakonec třetí případ, kdy $\dot{m} = \dot{m}_0$, dává fyzikální smysl.

```
def flow_plot(ax, gamma, mdot):
    xvals = np.logspace(-2, 2, 300)
    yvals = np.logspace(-5, 3, 300)
    n = len(xvals)
    X, Y = np.meshgrid(xvals, yvals)
    Z = np.array([[Psi(x, y, gamma, mdot) for x in xvals] for y in yvals])
    ax.contour(X, Y, Z, levels=[Psis(gamma, mdot)], colors=[blue])
    ax.contour(X, Y, Z, levels=[Psi0(gamma)], colors=[purple])
    ax.plot(xvals[2*n//3:], [yasy_inf(x, mdot) for x in xvals[2*n//3:]] ls=':', color=purple)
    ax.plot(xvals[:n//5], [yasy_0(x) for x in xvals[:n//5]], ls=':', color=purple)
    ax.set_xlabel(r'Radial coordinate $x$')
    ax.set_ylabel(r'Squared velocity $y$')
    ax.axhline(ys(gamma, mdot), ls='--', color='gray')
    ax.axvline(xs(gamma, mdot), ls='--', color='gray')
    ax.set_ylim(1e-5,1e+3)
    ax.set_xscale('log')
    ax.set_yscale('log')

fig, panels = plt.subplots(1, 3, figsize=(14,4))
flow_plot(panels[0], 4./3., 0.5*mdot0(4/3))
flow_plot(panels[1], 4./3., 1.5*mdot0(4/3))
flow_plot(panels[2], 4./3., mdot0(4/3))
panels[0].annotate(r'$\dot{m}<\dot{m}_0$', xy=(0.98, 0.98), xycoords="axes fraction", va="top", ha='right')
panels[1].annotate(r'$\dot{m}>\dot{m}_0$', xy=(0.98, 0.98), xycoords="axes fraction", va="top", ha='right')
panels[2].annotate(r'$\dot{m}=\dot{m}_0$', xy=(0.98, 0.98), xycoords="axes fraction", va="top", ha='right')

fig.tight_layout()
```

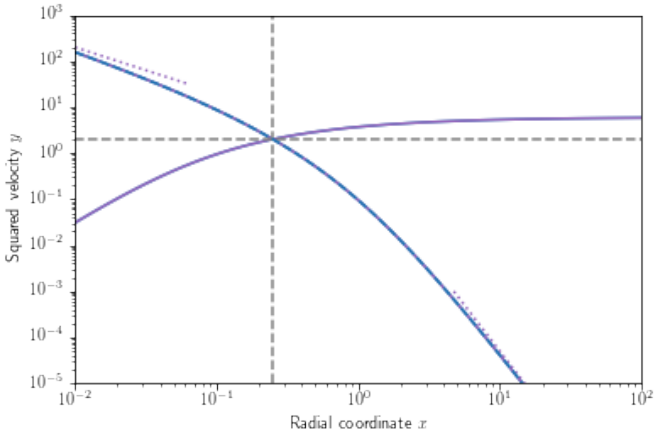


Obrázek 7.1: Tři případy pro různé hodnoty akrečního toku.

Je vhodné zavést funkci, která vrací transonické řešení y v daném bodě x – tady je: Hodnota kvadrátu rychlosti je nalezena jako řešení Bernoulliho rovnice pro dané x. Použijeme metodu bisekce. Jako okrajové hodnoty použijeme rychlost zvuku v sonickém bodě a dvě asymptotické formy (pro $x \to 0, \infty$) vynásobené „bezpečnostním“ faktorem

```
def exact_solution(x, gamma):
    mdot = mdot0(gamma)
    x0 = xs(gamma, mdot)
```

```
    y0 = ys(gamma, mdot)
    if x < x0:
        ya, yb = y0, max(yasy_0(x), 1e2*y0)
        return opt.bisect(lambda y: Psi(x, y, gamma, mdot)-Psi0(gamma), ya, yb)
    else:
        ya, yb = min(1e-2*yasy_inf(x, mdot), 1e-2*y0), y0
        return opt.bisect(lambda y: Psi(x, y, gamma, mdot)-Psi0(gamma), ya, yb)

fig, ax = plt.subplots(1)
flow_plot(ax, 4/3, mdot0(4/3))
xvals = np.logspace(-2, 2)
ax.plot(xvals, [exact_solution(x, 4/3) for x in xvals], ls='--', color=blue);
```



Obrázek 7.2: Přesné řešení pro kvadrát rychlosti.

7.4 Řešení s použitím relaxační metody, řešení problému vlastních hodnot na neznámé oblasti řešení

Chceme řešit Bondiho úlohu numericky. Nejprve si pojďme shrnout základní vlastnosti úlohy.

- Bondiho akrece je „okrajová úloha“. Řešení je vymezeno na velkých radiálních souřadnicích a navíc je další omezení dáno podmínkou regularity v sonickém bodě. Numerické řešení musí splňovat omezení na obou stranách oblasti řešení. Jak jsme viděli, *relaxační metoda* může být v tomto případě velmi nápomocná.
- V případě Bondiho akrece známe předem pozici sonického bodu, ale v obecném případě nemusí být jeho pozice známá. Abychom mohli lépe demonstrovat numerické řešení, budeme dále předstírat, že nevíme kde se sonický bod nachází. To však dále komplikuje situaci. Pokud bychom považovali bezrozměrnou radiální souřadnici za nezávislou proměnou, neznali bychom předem oblast řešení (nevěděli bychom, kde se nachází vnitřní okraj). Musíme tedy použít nějaký trik a tento problém vyřešit dříve než začneme přímo implementovat relaxační metodu.
- Protože je řešení dáno jednoduchou diferenciální rovnicí prvního řádu, je problém očividně příliš omezený a řešení existuje pouze pro určité hodnoty akrečního toku $\dot{m}$. To znamená, že Bondiho úloha je také problém vlastních hodnot. Hledáme konkrétní hodnotu parametru (bezrozměrného akrečního toku $\dot{m}$) tak, že existuje řešení kompatibilní s oběma okrajovými podmínkami. Jak můžeme použít relaxační metodu pro nalezení vlastních hodnot problému?

7.4.1 Shrnutí problému

Pojďme si nejprve shrnout problém v jeho matematické formě. Hledáme řešení $y(x)$ a hodnotu parametru $\dot{m}$ problému zadaného jednou obyčejnou diferenciální rovnicí,

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{\frac{4\tilde{a}^2}{x} - \frac{2}{x^2}}{1 - \frac{\tilde{a}^2}{y}}, \quad \tilde{a}^2 = \left(\frac{\dot{m}}{x^2 y^{1/2}}\right)^{\gamma-1}, \tag{7.24}$$

s dvěma okrajovými omezeními

$$y \approx \left(\frac{\dot{m}}{x^2}\right)^2 \quad (x \to \infty) \tag{7.25}$$

a

$$\frac{4\tilde{a}^2}{x_\mathrm{s}} - \frac{2}{x_\mathrm{s}^2} = 0 \quad \Leftrightarrow \quad y = \tilde{a}^2. \tag{7.26}$$

7.4.1.1 Úprava problému vlastních hodnot

Máme dvě okrajové podmínky pro jednu obyčejnou diferenciální rovnici, které závisí na jednom dodatečném parametru ($\dot{m}$). Hledáme řešení $y(x)$ a odpovídající hodnotu $\dot{m}$ takovou, že *obě* okrajové podmínky jsou splněny. Nejjednodušší způsob, jak najít řešení, je považovat parametr $\dot{m}$ za další závislou proměnou, $\dot{m} = \dot{m}(x)$, danou další obyčejnou diferenciální rovnicí $\mathrm{d}\dot{m}/\mathrm{d}x = 0$ zajišťující, že se bude jednat o konstantu. Pak dostaneme soustavu dvou obyčejných diferenciálních rovnic pro $y(x)$ a $\dot{m}(x)$ s dvěma okrajovými omezeními,

$$\frac{\mathrm{d}}{\mathrm{d}x}\begin{bmatrix}\dot{m}\\ y\end{bmatrix}=\begin{bmatrix}0\\ \left(\frac{4\tilde{a}^2}{x}-\frac{2}{x^2}\right)\left(1-\frac{\tilde{a}^2}{y}\right)^{-1}\end{bmatrix},\quad A\left(\dot{m},y\right)=\frac{4y}{x_a}-\frac{2}{x_a^2}=0,\quad B\left(\dot{m},y\right)=y-\frac{\dot{m}^2}{x_b^2}=0. \tag{7.27}$$

Pozice vnějšího okraje x_b je ve své podstatě volná, v praxi bychom měli volit rádius dostatečně daleko, tak aby skutečné řešení souhlasilo s asymptotickým tvarem, který jsme odvodili v předchozí části. Podíváme-li se na analytické řešení, vidíme, že už na $x_b \approx 10^2$ odpovídá skutečné řešení asymptotickému tvaru velmi dobře (alternativně můžeme zkusit najít další člen v asymptotickém rozvoji, který indikuje chybu předešlého řádu, a podle toho se rozhodnout, kam umístíme vnější okrajovou podmínku). Na druhou stranu, poloha vnitřní okrajové podmínky je stále neznámá – místo toho máme vztah

$$y(x_a)-\tilde{a}^2\left(\dot{m},x_a,y\right)=0. \tag{7.28}$$

To je mnohem lepší, ale stále ne perfektní. Pozice vnitřní okrajové podmínky x_a není předem známá, místo toho závisí na lokální hodnotě závislých proměnných y a $\dot{m}$.

7.4.1.2 Co udělat s vnitřní okrajovou podmínkou?

Vnitřní okrajová podmínka je očividně také neznámá, ale máme pro ní jedno omezení navíc. Co kdybychom do závislých proměnných zahrnuli také radiální souřadnici? Jinak řečeno, zavedeme novou závislou proměnnou q, tak že $x = x(q)$, $y(x) \equiv y[x(q)] = y(q)$, a obdobně pro $\dot{m}$. Pak lze derivaci podle x rozložit na derivaci složené funkce

$$\frac{\mathrm{d}}{\mathrm{d}x}\equiv\frac{\mathrm{d}q}{\mathrm{d}x}\frac{\mathrm{d}}{\mathrm{d}q} \tag{7.29}$$

Pojďme se podívat, co se stane s naším systémem rovnic. Nyní máme tři neznámé funkce $\dot{m}(q)$, $x(q)$ a $y(q)$ a systém obyčejných diferenciálních rovnici vypadá takto:

$$\frac{\mathrm{d}}{\mathrm{d}q}\begin{bmatrix}\dot{m}\\ x\\ y\end{bmatrix}=\begin{bmatrix}0\\ \psi\\ \psi\left(\frac{4\tilde{a}^2}{x}-\frac{2}{x^2}\right)\left(1-\frac{\tilde{a}^2}{y}\right)^{-1}\end{bmatrix}. \tag{7.30}$$

Objevuje se zde nová funkce $\psi(q) = \mathrm{d}x/\mathrm{d}q$, která vychází ze změny nezávislé proměnné x na q. Tato funkce je víceméně volitelná, pokud zvolíme $\psi \equiv 1$, dostaneme původní problém. Pokud jí zvolíme rovnou nějaké jiné konstantě (např. α), dostaneme jen jiné škálování radiální souřadnice ($q = x/\alpha$). Na první pohled se zdá, že jsme nic nevylepšili: měřit radiální souřadnici v x nebo v násobcích x – jaký je v tom rozdíl? Avšak to je právě smysl této práce. Všimněte si, že stále nevíme kde se nachází vnitřní okrajová podmínka x_a, avšak, jakákoli je její hodnota, vždy bude existovat taková hodnota ψ, která bude transformovat (neznámý) interval x, $[x_a, x_b]$ do daného předepsaného intervalu q, třeba $[0, 1]$. Zaveďme $x(0) = x_a$ a $x(1) = x_b$. Hodnota ψ, která definuje tuto transformaci je neznámá, ale můžeme jí považovat za další vlastní podmínku problému, úplně stejně jako jsme to udělali s $\dot{m}$. Takže, v konečném důsledku řešíme problém pro čtyři závislé proměnné $\dot{m}$, x, y a ψ danými rovnicemi

$$\frac{\mathrm{d}}{\mathrm{d}q}\begin{bmatrix}\dot{m}\\ x\\ y\\ \psi\end{bmatrix}=\begin{bmatrix}0\\ \psi\\ \psi\left(\frac{4\tilde{a}^2}{x}-\frac{2}{x^2}\right)\left(1-\frac{\tilde{a}^2}{y}\right)^{-1}\\ 0\end{bmatrix},\quad \tilde{a}^2\left(\dot{m},x,y\right)=\left(\frac{\dot{m}}{x^2y^{1/2}}\right)^{\gamma-1}. \tag{7.31}$$

s dvěma omezeními řešení na obou stranách,

$$A\left([\dot{m},x,y,\psi]^T\right)=\begin{bmatrix}\frac{4\tilde{a}^2}{x}-\frac{2}{x^2}\\ y-\tilde{a}^2\end{bmatrix}=0,\quad B\left([\dot{m},x,y,\psi]^T\right)=\begin{bmatrix}x-x_b\\ y-\frac{\dot{m}^2}{x^4}\end{bmatrix}=0 \tag{7.32}$$

V této formě je již možné problém řešit použitím relaxační metody. Avšak než to uděláme, musíme přidat několik poznámek.

7.4.1.3 Obecná poznámka k síti souřadnic

Transformace z neznámého intervalu $[x_a,x_b]$ na pevný interval v q:
V předchozím textu jsme předpokládali, že $[x_a,x_b]\to[0,1]$, takže $x(q=0)=x_a$ a $x(q=1)=x_b$. Ve skutečnosti je mnohem obvyklejší transformace $[x_a,x_b]\to[0,M-1]$, kde M je počet bodů sítě. Pak jednotlivé body sítě odpovídají celočíselným hodnotám nezávislé proměnné q (např. $x_i\equiv x(q=i)$). Dosud jsme předpokládali lineární transformaci $q\to x$ takovou, že $x(0)=x_a$ a $x(M-1)=x_b$. Takže transformace má tvar

$$x=x_a+\frac{q}{M-1}\left(x_b-x_a\right),\quad \psi=\frac{\mathrm{d}x}{\mathrm{d}q}=\frac{x_b-x_a}{M-1} \tag{7.33}$$

Můžeme ale také zvolit nelineární funkci. Například pokud vezmeme místo $\mathrm{d}x/\mathrm{d}q=\psi$,

$$\frac{\mathrm{d}x}{\mathrm{d}q}=\psi\theta(x), \tag{7.34}$$

kde ψ je konstanta a $\theta(x)$ je funkcí x. Po integraci pak dostaneme

$$\int\frac{\mathrm{d}x}{\theta(x)}=\psi q+\mathrm{const} \tag{7.35}$$

Funkce θ očividně ovlivní transformaci $q\to x$. Jelikož celočíslené hodnoty q označují jednotlivé body sítě, $\theta(x)$ určuje umístění těchto bodů. Například volba $\theta(x)=x$ definuje logaritmické rozložení bodů sítě, které bude využito při implementaci dále.

Můžeme použít i pokročilejší rozložení bodů, například metodu adaptivní sítě. Například

$$\theta(x,Y)=\Delta+\beta Y\left(\frac{\mathrm{d}Y}{\mathrm{d}x}\right)^{-1} \tag{7.36}$$

nám dává uniformní síť, kde je vzdálenost mezi body rovná Δ právě tehdy, když $\beta=0$. Pro nenulové β je rozložení bodů ovlivněno také magnitudou funkční derivace: síť je hustší v místech, kde se řešení mění rychleji.

7.5 Implementace

Při implementaci využijeme relaxační metodu pro řešení Bondiho akrece. Použijeme 4 proměnné $[\dot{m},x,y,\psi]^T$. Dvě z nich, $\dot{m}$ a ψ jsou vlastní hodnoty problému. Pro transformaci $q\to x$ použijeme $\mathrm{d}x/\mathrm{d}q=x\psi$, tedy logaritmické rozložení bodů sítě. Hodnota $q=0$ odpovídá vnějšímu okraji umístěnému na $x=10^2$. Vnitřní okraj je dán funkcí $q=M-1$, kde M je počet bodů sítě. Voláme relaxační metodu implementovanou v `relaxation.py` (viz předchozí kapitola). Jako počáteční odhad použijeme dvě vlastní hodnoty $\dot{m}$ a ψ. Poté integrujeme obyčejné diferenciální rovnice použitím metody Runge-Kutha až do bodu, kdy rychlost dosáhne 0.8 lokální rychlosti zvuku. S tímto řešením můžeme začít následný relaxační proces.

Jelikož relaxační metoda vyžaduje implementaci jakobiánu pravé strany rovnic a hraniční omezení, musíme buďto najít přesný vztah analyticky, nebo jej vypočítat numericky – tak, jak to uděláme dále. Dvě funkce `numder` a `numjac` berou funkci jedné nebo více proměnných a vrací funkci, která počítá numerickou derivaci metodou centrální diference. Elegantnější řešení by však bylo využít automatickou diferenciaci.

```
def numder(f, dx=1e-10):
    return lambda x: (f(x+dx) - f(x-dx))/(2*dx)

def numjac(F, eps=1e-5):
    def jac(X):
        n = len(X)
        result = []
        for i in range(n):
            FF = lambda t: F(np.concatenate((X[:i], [t], X[i+1:]), axis=None))
            result.append((FF(X[i]+eps) - FF(X[i]-eps))/(2*eps))
        return np.transpose(np.array(result))
    return jac

# example: der is derivative of the function x --> x^2
der = numder(lambda x: x**2)
print(der(1))

# example: jac is jacobian of the function [x, y] --> [x^2, x+y]
jac = numjac(lambda X: np.array([X[0]**2, X[0]+X[1]]))
print(jac([1,1]))
```

Zde je implementace:

```
import scipy.integrate as integrate
import relaxation

class BondiFlow:
    """
    Solution class for finding regular solutions of the Bondi equation

    Initialization:
    ---------------
    gamma : polytropic index of the flow
    mdot  : initial guess of the accretion rate
    xinf  : position of the outer boundary
    M     : rough number of the grid points (logarithmically spaced)
    """

    def __init__(self, gamma, mdot, xinf, M):
        """ Initialization. """
        # set the private variables:
        self._gamma = gamma
        self._mdot = mdot
        self._xinf = xinf
        # set the Jacobians of RHS and boundary constraints:
        # Note since F depends on both dependent and independent variables (q and X),
```

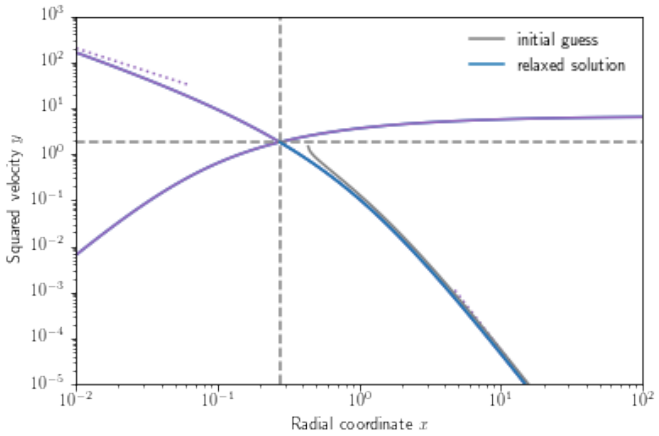
```
24          # and Jac is related only to X, we have to do the following trick:
25          self.jacF = lambda q, X: numjac(lambda Y: self.F(0, Y))(X)
26          self.jacA = numjac(self.A)
27          self.jacB = numjac(self.B)
28          # finally, calculate the initial guess:
29          self.make_initial_guess(M)
30
31      def make_initial_guess(self, M):
32          """ Calculating the initial guess using direct RK integration. """
33          # initial values of the solution variables = first 4 entry of the solution vector _Ys
34          # (here psi0 is calculated by integrating dx/dq equation and assuming that the
35          # sonic point is approximately at xs ~ 1/2)
36          xs = 0.5
37          psi0 = math.log(xs/self._xinf)/(M-1)
38          self._Ys = np.array([self._mdot, self._xinf, math.log(self._mdot**2/self._xinf**4), psi0])
39          self._qs = [0]
40
41          # introduce the scipy.integrate ode solver:
42          solver = integrate.ode(self.F).set_integrator('vode', method='bdf', rtol=1e-10, nsteps=10000)
43          solver.set_initial_value(self._Ys, self._qs[0])
44
45          # integrate until we the velocity is greater than sqrt(0.8) sound speed or number of
46          # mesh point exceeds 1.5 times the initially setted. The local values are storred in
47          # the local solution vectors _qs and _Ys:
48          q = 1
49          finish = False
50          while not(finish):
51              Y = solver.integrate(q)
52              self._qs.append(q)
53              self._Ys = np.concatenate((self._Ys, Y))
54              q += 1
55              [mdot, x, y, psi] = Y
56              a2 = (mdot/(x*x*math.exp(y/2)))**(self._gamma - 1)
57              finish = (q > 1.5*M) or (math.exp(y) > 0.8*a2)
58
59      def F(self, q, Y):
60          """ The right-hand sides of the ODEs. """
61          [mdot, x, y, psi] = Y
62          a2 = (mdot/(x*x*math.exp(y/2)))**(self._gamma - 1)
63          f = (4*a2/x - 2/x**2)/(math.exp(y) - a2)
64          return np.array([0, x*psi, f*x*psi, 0], float)
65
66      def A(self, Y):
67          """ Constraints at x_inf. (corresponding to q=0) """
68          [mdot, x, y, psi] = Y
69          xinf = 1e2
70          return np.array([x - xinf, math.exp(y) - mdot**2/x**4], float)
71
72      def B(self, Y):
73          """ Constraints at the sonic point (corresponding to q=M-1). """
74          [mdot, x, y, psi] = Y
75          a2 = (mdot/(x*x*math.exp(y/2)))**(self._gamma - 1)
76          return np.array([math.exp(y) - a2, 4*a2/x - 2/x**2], float)
```

```
77
78      def calculate(self, k=1., atol=1e-3, maxiter=50):
79          """
80          Iterative use of the relaxation method.
81          Returns (success, i), where success is True if the iteration converged and
82          i is number of the iteration steps.
83          """
84          success = False
85          i = 1
86          # The iterative process continues until number of iteration exceeds `maxiter`
87          # or the L2-norm of the improvements < 'atol'
88          while not(success) and (i < maxiter):
89              dY = k*relaxation.relaxation_step(self._qs, self._Ys,
90                                                self.F, self.jacF, self.A, self.jacA, self.B, self.jacB)
91              self._Ys += dY
92              self._mdot += dY[0]
93              i += 1
94              success = (np.linalg.norm(self.B(self._Ys[-4:])) < atol)
95          return success, i
96
97      def modify_gamma(self, gamma):
98          """ Change the value of polytropic index. """
99          self._gamma = gamma
100
101     @property
102     def mdot(self):
103         """ Actual accretion rate. """
104         return self._mdot
105
106     @property
107     def gamma(self):
108         """ Actual polytropic index. """
109         return self._gamma
110
111     @property
112     def xs(self):
113         """ Actual position of the sonic point. """
114         return self._Ys[-3]
115
116     def plot_solution_xy(self, ax, **kwargs):
117         """ Place the solution into the axes object ax. """
118         ax.plot(self._Ys[1::4], np.exp(self._Ys[2::4]), **kwargs)
```

... a příklad použití. Zkoušíme najít řešení pro $\gamma = 1.3$ s počátečním odhadem překračujícím správnou hodnotu o 10%.

```
1   # set the params and initialize the solver:
2   gamma = 1.3
3   mdot_guess = 1.1*mdot0(gamma)
4   bondi = BondiFlow(gamma=gamma, mdot=mdot_guess, xinf=100., M=100)
5
6   # plot the initial guess (the relaxation is not called yet):
```

```
fig, ax = plt.subplots(1)
flow_plot(ax, bondi.gamma, mdot0(bondi.gamma))
bondi.plot_solution_xy(ax, ls='-', color='gray', marker='', label='initial guess')

# make the relaxation process:
success, i = bondi.calculate(atol=1e-3, maxiter=100, k=0.2)

# plot the solution:
bondi.plot_solution_xy(ax, ls='-', color=blue, marker='', label='relaxed solution')
ax.legend()

# print the report:
print('success: {}, number of iterations: {}'.format(success, i))
print('initial mdot = {}'.format(mdot_guess))
print('mdot found   = {}'.format(bondi.mdot))
print('correct mdot = {}'.format(mdot0(gamma)))
```



Obrázek 7.3: Numerické řešení

Úkol

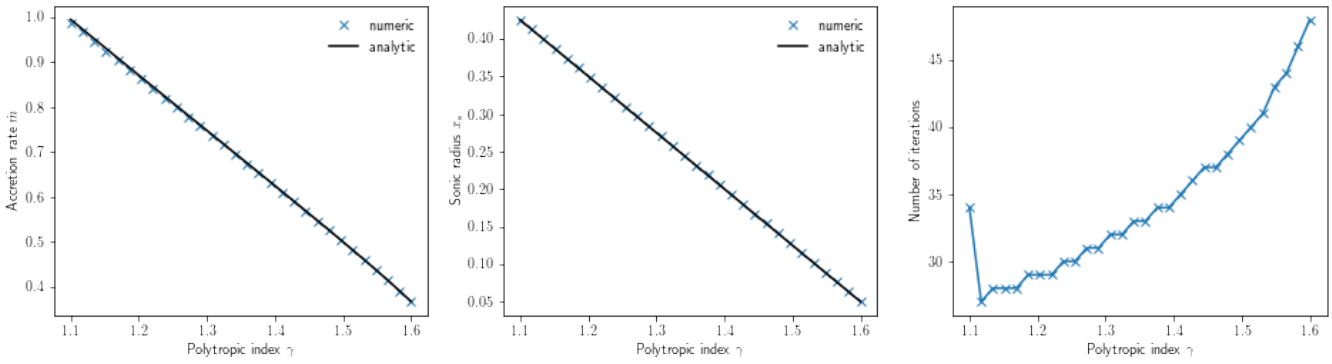
Zkuste implementovat řešení pomocí adaptivní sítě bodů.

7.5.1 Zkoumání prostoru parametrů

Relaxační metoda je úžasný nástroj pro průzkum prostoru parametrů. Jediný volný parametr je polytropický index γ. Pokud jiné řešení začneme s počáteční hodnotou γ_0, zjistíme, že řešení stejně jako předtím začíná s počátečním odhadem. Poté změníme poltropický index z γ_0 na $\gamma_0 + \mathrm{d}\gamma$. A použijeme předchozí řešení jako počáteční odhad pro nové řešení. Tato procedura je implementována níže:

```
# various values of the polytropic index:
gammas = np.linspace(1.1, 1.6, 30)

# initial guess for the first solution, and relaxed solution:
mdot_guess = 1.1*mdot0(gammas[0])
```

```
bondi = BondiFlow(gamma=gammas[0], mdot=mdot_guess, xinf=100, M=100)
sucess, i = bondi.calculate(atol=1e-3, maxiter=100, k=0.2)

# we store the accretion rate, position of the sonic point and number of iterations:
mdots = [bondi.mdot]
xss = [bondi.xs]
iters = [i]

# now we go through all values of the polytropic index, modify the solver
# and upadate & store the solutions:
for gamma in gammas[1:]:
    bondi.modify_gamma(gamma)
    sucess, i = bondi.calculate(atol=1e-3, maxiter=100, k=0.2)
    mdots.append(bondi.mdot)
    xss.append(bondi.xs)
    iters.append(i)

# finally, we plot the resuts. Black solid line is the solution of the analytic theory:
fig, panels = plt.subplots(1,3, figsize=(16,4))
panels[0].plot(gammas, mdots, ls='', marker='x', label='numeric')
panels[0].plot(gammas, [mdot0(gamma) for gamma in gammas], ls='-', color='black', label='analytic')
panels[0].set_xlabel(r'Polytropic index $\gamma$')
panels[0].set_ylabel(r'Accretion rate $\dot{m}$')
panels[0].legend()

panels[1].plot(gammas, xss, ls='', marker='x', label='numeric')
panels[1].plot(gammas, [xs(gamma, mdot0(gamma)) for gamma in gammas], ls='-', color='black', label='analytic')
panels[1].set_xlabel(r'Polytropic index $\gamma$')
panels[1].set_ylabel(r'Sonic radius $x_\mathrm{s}$')
panels[1].legend()

panels[2].plot(gammas, iters, ls='-', marker='x')
panels[2].set_xlabel(r'Polytropic index $\gamma$')
panels[2].set_ylabel(r'Number of iterations');
```



Souhrn kapitoly

V této kapitole jsme implementovali problém Bondiho akrece a vyřešili jej pomocí relaxační metody. Také jsme zavedli různé sítě bodů, od lineární, přes logaritmickou až po adaptivní.

Otázky

1. Jaký má význam rovnice kontinuity?
2. Co reprezentuje konstanta $\dot{m}$?
3. Jak se změní řešení při použití jiného polytropického indexu?
4. Jakým způsobem lze určit vnější okrajovou podmínku pro radiální souřadnici?
5. Jaká je výhoda použití logaritmické sítě oproti lineární?

Kapitola 8

Některé aproximační metody řešení diferenciálních rovnic

Martin Blaschke

Rychlý náhled: V této kapitole si představíme aproximační řešení diferenciálních rovnic, konkretně Poissonovu a Linstedtovu metodu a především Krylovu-Bogoliubovu metodu. Implementace je nastíněna v programu `Wolfram Mathematica`. Kapitola je určená pro pokročilé studenty.

Klíčová slova: numerické řešení diferenciálních rovnic, aproximační metody, Poissonova metoda, Linstedtova metoda, Krylova-Bogoliubova metoda, Wolfram Mathematica

Cíle studia: Cílem této kapitoly je ukázat aproximativní řešení diferenciálních rovnic, která jsou užitečná v případě, kdy nelineární členy jsou malé (zanedbatelné) vzhledem k lineární části diferenciální rovnice. Po prostudování kapitoly bude student schopen použít tuto metodu na řešení diferenciálních rovnic.

8.1 Řešení v podobě Talorovy řady generované v Mathematice

Než přejdeme k perturbační metodě, ukážeme si, že program Mathematica je schopen generovat řešení ve tvaru Taylorovy řady, které bude validní na určitém rozpětí parametrů.

Abychom zjistili, jak lze program `Mathematica` použít k vygenerování Taylorovy řady, uvažujeme o konkrétním příkladu jednoduchého (to znamená $g = l$) matematického kyvadla: $\theta'' + \sin\theta = 0$ s počátečním úhlem $\theta(0) = \pi/3$ a počáteční rychlostí $\theta'(0) = 0$.

Řešený příklad

Odvozte řešení Taylorovy řady jednoduchého matematického kyvadla podle dané počáteční podmínky, do osmého řádu (t^8) přesnosti.

Řešení

Začneme tím, že rozvineme θ[t] v Taylorovu řadu kolem času t = 0,
přičemž řády vyšší než t^8 budeme ignorovat a vložíme počáteční podmínky do řady .

```
Clear["Global`*"]
s1[t_] = Series[θ[t], {t, 0, 8}] /. {θ[0] → π / 3, θ'[0] → 0}
```

$$\frac{\pi}{3} + \frac{1}{2}\,\theta''[0]\,t^2 + \frac{1}{6}\,\theta^{(3)}[0]\,t^3 + \frac{1}{24}\,\theta^{(4)}[0]\,t^4 + \frac{1}{120}\,\theta^{(5)}[0]\,t^5 + \frac{1}{720}\,\theta^{(6)}[0]\,t^6 + \frac{\theta^{(7)}[0]\,t^7}{5040} + \frac{\theta^{(8)}[0]\,t^8}{40\,320} + O[t]^9$$

Toto následně vložíme do diferenciální rovnice a automaticky seřadíme jednotlivé členy podle řády v proměnné t. Toto seřazení zajistí příkaz `Simplify[]`, nebo v některých případech i příkaz `FullSimplify[]`.

```
ode = Simplify[D[s1[t], {t, 2}] + Sin[s1[t]] == 0]
```

$$\left(\frac{\sqrt{3}}{2} + \theta''[0]\right) + \theta^{(3)}[0]\,t + \frac{1}{4}\left(\theta''[0] + 2\,\theta^{(4)}[0]\right)t^2 + \frac{1}{12}\left(\theta^{(3)}[0] + 2\,\theta^{(5)}[0]\right)t^3 + \frac{1}{48}\left(-3\sqrt{3}\,\theta''[0]^2 + \theta^{(4)}[0] + 2\,\theta^{(6)}[0]\right)t^4 + \frac{1}{240}\left(-10\sqrt{3}\,\theta''[0]\,\theta^{(3)}[0] + \theta^{(5)}[0] + 2\,\theta^{(7)}[0]\right)t^5 + \frac{\left(-10\sqrt{3}\,\theta^{(3)}[0]^2 - 15\left(\theta''[0]^3 + \sqrt{3}\,\theta''[0]\,\theta^{(4)}[0]\right) + \theta^{(6)}[0] + 2\,\theta^{(8)}[0]\right)t^6}{1440} + O[t]^7 == 0$$

Jelikož proměnná t (čas) může nabývat libovolných hodnot, je jediným způsobem jak zajisti obecné řešení to, že všechny koeficienty ve všech řádech t jsou nulové. Tato podmínka může být vyjádřena pomocí příkazu `LogicalExpand[]`.

```
eq1 = LogicalExpand[ode]
```

$$\frac{\sqrt{3}}{2} + \theta''[0] == 0 \;\&\&\; \theta^{(3)}[0] == 0 \;\&\&\; \frac{1}{4}\left(\theta''[0] + 2\,\theta^{(4)}[0]\right) == 0 \;\&\&\; \frac{1}{12}\left(\theta^{(3)}[0] + 2\,\theta^{(5)}[0]\right) == 0 \;\&\&\; \frac{1}{48}\left(-3\sqrt{3}\,\theta''[0]^2 + \theta^{(4)}[0] + 2\,\theta^{(6)}[0]\right) == 0 \;\&\&\; \frac{1}{240}\left(-10\sqrt{3}\,\theta''[0]\,\theta^{(3)}[0] + \theta^{(5)}[0] + 2\,\theta^{(7)}[0]\right) == 0 \;\&\&\; \frac{-10\sqrt{3}\,\theta^{(3)}[0]^2 - 15\left(\theta''[0]^3 + \sqrt{3}\,\theta''[0]\,\theta^{(4)}[0]\right) + \theta^{(6)}[0] + 2\,\theta^{(8)}[0]}{1440} == 0$$

Symbol `&&` objevující se mezi rovnicemi reprezentuje logický součet. Tento systém obyčejných rovnic můžeme nyní přímo vyřešit.

```
Solve[eq1]
```

$$\left\{\left\{\theta''[0] \to -\frac{\sqrt{3}}{2},\ \theta^{(3)}[0] \to 0,\ \theta^{(4)}[0] \to \frac{\sqrt{3}}{4},\ \theta^{(5)}[0] \to 0,\ \theta^{(6)}[0] \to \sqrt{3},\ \theta^{(7)}[0] \to 0,\ \theta^{(8)}[0] \to -\frac{49\sqrt{3}}{8}\right\}\right\}$$

Řešení ve formě řady pak nalezneme přímím použitím jednotlivých výsledků.

```
sol = s1[t] /. %[[1]]
```

$$\frac{\pi}{3} - \frac{\sqrt{3}\,t^2}{4} + \frac{t^4}{32\sqrt{3}} + \frac{t^6}{240\sqrt{3}} - \frac{7\,t^8}{15\,360\sqrt{3}} + O[t]^9$$

Povšimněme si, že vyjádření neobsahuje liché členy řady. Ke konečnému vyjádření výsledku můžeme použít příkazu `Normal[]` k odstranění symbolu reprezentujícího vyšší řády přesnosti.

```
Normal[sol]
```

$$\frac{\pi}{3} - \frac{\sqrt{3}\,t^2}{4} + \frac{t^4}{32\sqrt{3}} + \frac{t^6}{240\sqrt{3}} - \frac{7\,t^8}{15\,360\sqrt{3}}$$

Toto řešení ve tvaru řady bude přesné (lépe řečeno užitečné) pouze pro omezené množství času t, o čemž se v tomto případě lze přesvědčit porovnáním s analytickým řešením.

Úkoly

1. Spočítejte řešení příkladu až do $O(t^{15})$ a prodiskutujete možnost a praktičnost nalezení tohoto řešení metodou papír - tužka.
2. Vykreslete (`Plot[]`) řešení pomocí řady pro různé řády přesnosti vzhledem k analytickému řešení.
3. Nahraďte $\sin(x)$ členem $(x^2+x3)\cos(x)$ a proveďte celý výpočet znovu. Opět uvažujte do jaké míry by tento postup byl užiteční pokud by byl prováděn metodou papír - tužka.

8.2 Poissonova metoda

Perturbační metody jsou použitelné, když je nultý řád analyticky řešitelný a vyšší řády jsou malé. Tuto malost zajišťuje parametr ϵ, který je do problému vložen ručně. Uvažujme například *Van der Polovu rovnici*

```
D[x[t], {t, 2}] - ϵ (1 - x[t]^2) D[x[t], t] + x[t] == 0
```

$$x[t] - \epsilon\left(1 - x[t]^2\right) x'[t] + x''[t] == 0$$

Parametr byl vložen před závorku tak, aby nultý řád ($\epsilon = 0$) byla jednoduchá lineární diferenciální rovnice.

Základní myšlenka perturbačních metod spočívá v tom, že i řešení se dá rozepsat pomocí mocninné řady v ϵ (ne nutně Taylorovi).
Jestliže je ϵ skutečně malé, obyčejně potřebujeme pouze několik členů řady k nalezení přibližného řešení dostatečné přesnosti.
Mocninné řady obvykle divergují, a proto je občas potřeba užít dalších metod, které extrahují užitečnou informaci z divergující řady (*Padé aproximace*).

Řešený příklad

Nalezněte řešení do 6-tého řádu v ϵ pomocí Poissonovy perturbační metody pro

```
x[τ] + ϵ x[τ]^2 + x'[τ] == 0
```

Řešení

```
Clear["Global`*"]
n = 6;
x[τ] = Sum[xᵢ[τ] ϵ^i, {i, 0, n}]
```

$$x_0[\tau] + \epsilon\, x_1[\tau] + \epsilon^2 x_2[\tau] + \epsilon^3 x_3[\tau] + \epsilon^4 x_4[\tau] + \epsilon^5 x_5[\tau] + \epsilon^6 x_6[\tau]$$

Pozn. Index se vytvoří kombinací kláves Ctrl + –. Nelineární rovnice je poté rozvedena

```
eq = Expand[x[τ] + ϵ x[τ]^2 + D[x[τ], τ]]
```

```
x0[τ] + ϵ x0[τ]^2 + ϵ x1[τ] + 2 ϵ^2 x0[τ] x1[τ] + ϵ^3 x1[τ]^2 + ϵ^2 x2[τ] +
 2 ϵ^3 x0[τ] x2[τ] + 2 ϵ^4 x1[τ] x2[τ] + ϵ^5 x2[τ]^2 + ϵ^3 x3[τ] + 2 ϵ^4 x0[τ] x3[τ] +
 2 ϵ^5 x1[τ] x3[τ] + 2 ϵ^6 x2[τ] x3[τ] + ϵ^7 x3[τ]^2 + ϵ^4 x4[τ] + 2 ϵ^5 x0[τ] x4[τ] +
 2 ϵ^6 x1[τ] x4[τ] + 2 ϵ^7 x2[τ] x4[τ] + 2 ϵ^8 x3[τ] x4[τ] + ϵ^9 x4[τ]^2 + ϵ^5 x5[τ] +
 2 ϵ^6 x0[τ] x5[τ] + 2 ϵ^7 x1[τ] x5[τ] + 2 ϵ^8 x2[τ] x5[τ] + 2 ϵ^9 x3[τ] x5[τ] +
 2 ϵ^10 x4[τ] x5[τ] + ϵ^11 x5[τ]^2 + ϵ^6 x6[τ] + 2 ϵ^7 x0[τ] x6[τ] + 2 ϵ^8 x1[τ] x6[τ] +
 2 ϵ^9 x2[τ] x6[τ] + 2 ϵ^10 x3[τ] x6[τ] + 2 ϵ^11 x4[τ] x6[τ] + 2 ϵ^12 x5[τ] x6[τ] +
 ϵ^13 x6[τ]^2 + x0'[τ] + ϵ x1'[τ] + ϵ^2 x2'[τ] + ϵ^3 x3'[τ] + ϵ^4 x4'[τ] + ϵ^5 x5'[τ] + ϵ^6 x6'[τ]
```

A jednotlivé koeficienty jsou dány dohromady do eq2.

```
eq2 = Collect[eq, ϵ]
```

```
x0[τ] + 2 ϵ^12 x5[τ] x6[τ] + ϵ^13 x6[τ]^2 +
 ϵ^7 (x3[τ]^2 + 2 x2[τ] x4[τ] + 2 x1[τ] x5[τ] + 2 x0[τ] x6[τ]) +
 ϵ^8 (2 x3[τ] x4[τ] + 2 x2[τ] x5[τ] + 2 x1[τ] x6[τ]) +
 ϵ^9 (x4[τ]^2 + 2 x3[τ] x5[τ] + 2 x2[τ] x6[τ]) + ϵ^10 (2 x4[τ] x5[τ] + 2 x3[τ] x6[τ]) +
 ϵ^11 (x5[τ]^2 + 2 x4[τ] x6[τ]) + x0'[τ] + ϵ (x0[τ]^2 + x1[τ] + x1'[τ]) +
 ϵ^2 (2 x0[τ] x1[τ] + x2[τ] + x2'[τ]) + ϵ^3 (x1[τ]^2 + 2 x0[τ] x2[τ] + x3[τ] + x3'[τ]) +
 ϵ^4 (2 x1[τ] x2[τ] + 2 x0[τ] x3[τ] + x4[τ] + x4'[τ]) +
 ϵ^5 (x2[τ]^2 + 2 x1[τ] x3[τ] + 2 x0[τ] x4[τ] + x5[τ] + x5'[τ]) +
 ϵ^6 (2 x2[τ] x3[τ] + 2 x1[τ] x4[τ] + 2 x0[τ] x5[τ] + x6[τ] + x6'[τ])
```

Protože ϵ je libovolný parametr musí být v řešení každý koeficient roven nule. Příkaz `CoefficientList[]` nám udává seznam koeficientů, které je vhodné zobrazit v tabulce.

```
Table[CoefficientList[eq2, ϵ][[i]] == 0, {i, 1, n + 1}] // TableForm
```

```
x0[τ] + x0'[τ] == 0
x0[τ]^2 + x1[τ] + x1'[τ] == 0
2 x0[τ] x1[τ] + x2[τ] + x2'[τ] == 0
x1[τ]^2 + 2 x0[τ] x2[τ] + x3[τ] + x3'[τ] == 0
2 x1[τ] x2[τ] + 2 x0[τ] x3[τ] + x4[τ] + x4'[τ] == 0
x2[τ]^2 + 2 x1[τ] x3[τ] + 2 x0[τ] x4[τ] + x5[τ] + x5'[τ] == 0
2 x2[τ] x3[τ] + 2 x1[τ] x4[τ] + 2 x0[τ] x5[τ] + x6[τ] + x6'[τ] == 0
```

```
sol0 = Flatten[DSolve[{CoefficientList[eq2, ϵ][[1]] == 0, x0[0] == 1}, x0[τ], τ]]
```

```
{x0[τ] → e^-τ}
```

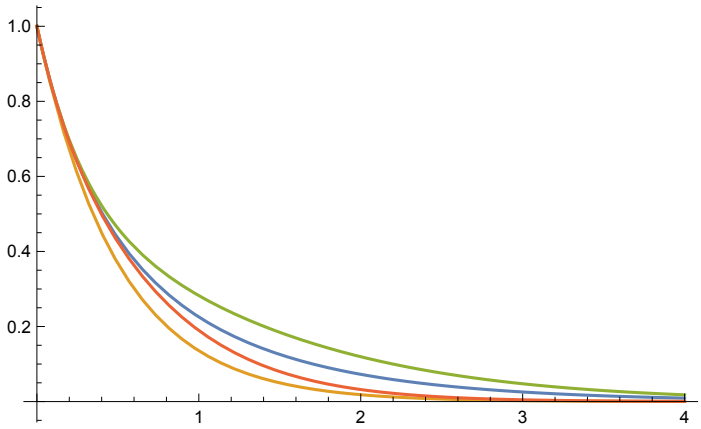
```
sol1 =
 Flatten[DSolve[{CoefficientList[eq2, ϵ][[2]] == 0 /. sol0, x1[0] == 0}, x1[τ], τ]]
sol2 = Flatten[
  DSolve[{CoefficientList[eq2, ϵ][[3]] == 0 /. sol0 /. sol1, x2[0] == 0}, x2[τ], τ]]
sol3 = Flatten[DSolve[{CoefficientList[eq2, ϵ][[4]] == 0 /. sol0 /. sol1 /. sol2,
    x3[0] == 0}, x3[τ], τ]]
sol4 = Flatten[DSolve[{CoefficientList[eq2, ϵ][[5]] == 0 /. sol0 /. sol1 /. sol2 /.
     sol3, x4[0] == 0}, x4[τ], τ]]
sol5 = Flatten[DSolve[{CoefficientList[eq2, ϵ][[6]] == 0 /. sol0 /. sol1 /. sol2 /.
      sol3 /. sol4, x5[0] == 0}, x5[τ], τ]]
sol6 = Flatten[DSolve[{CoefficientList[eq2, ϵ][[7]] == 0 /. sol0 /. sol1 /. sol2 /.
       sol3 /. sol4 /. sol5, x6[0] == 0}, x6[τ], τ]]
```

```
{x1[τ] → -e^-2 τ (-1 + e^τ)}
{x2[τ] → e^-3 τ (-1 + e^τ)^2}
{x3[τ] → -e^-4 τ (-1 + e^τ)^3}
{x4[τ] → e^-5 τ (-1 + e^τ)^4}
```

```
{x5[τ] → -e^(-6 τ) (-1 + e^τ)^5}
{x6[τ] → e^(-7 τ) (-1 + e^τ)^6}
Simplify[x[τ] /. sol0 /. sol1 /. sol2 /. sol3 /. sol4 /. sol5 /. sol6]
e^(-7 τ) (e^(6 τ) - e^(5 τ) (-1 + e^τ) ϵ + e^(4 τ) (-1 + e^τ)^2 ϵ^2 -
    e^(3 τ) (-1 + e^τ)^3 ϵ^3 + e^(2 τ) (-1 + e^τ)^4 ϵ^4 - e^τ (-1 + e^τ)^5 ϵ^5 + (-1 + e^τ)^6 ϵ^6)
GeneralSol = FullSimplify[ Sum[(-ϵ)^k e^(-(k+1) τ) (-1 + e^τ)^(k), {k, 0, Infinity}]]
        1
-------------------
-ϵ + e^τ (1 + ϵ)
```

V konečném kroku, můžeme jednotlivé řády perturbativního řešení porovnat s analytickým výsledkem. Zde se obvykle ϵ pokládá hodnotě 1.

```
Plot[{1/(-ϵ + e^τ (1 + ϵ)) /. ϵ → 1, Sum[(-ϵ)^k e^(-(k+1) τ) (-1 + e^τ)^(k), {k, 0, 1}] /. ϵ → 1,
  Sum[(-ϵ)^k e^(-(k+1) τ) (-1 + e^τ)^(k), {k, 0, 2}] /. ϵ → 1,
  Sum[(-ϵ)^k e^(-(k+1) τ) (-1 + e^τ)^(k), {k, 0, 3}] /. ϵ → 1}, {τ, 0, 4}]
```



Úkoly

1. Nalezněte perturbační řešení pro nelineární tvrdou strunu danou rovnicí:

   ```
   x''[τ] + ω^2 x[τ] + x^3[τ] == 0
   ```

2. Nalezněte Poissonovu expanzi do desátého řádu v ϵ pro ODE:

   ```
   x'[τ] + x[τ] + x^4[τ] == 0
   x'[τ] - x[τ] + x^5[τ] == 0
   x'[τ] - x[τ] - x^4[τ] == 0
   ```

8.3 Lindstedtova metoda (Poincarého-Lindstedtova metoda)

V poruchové teorii je Lindstedtova metoda (technika) užitá pro uniformní aproximaci periodických řešení obyčejných dif. rovnic (ODE), když ostatní metody selhávají.

Demonstrujme metodu na jednoduchém příkladu tvrdé nelineární struny. Abychom mohli vyřešit rovnici

```
x''[t] + ω^2 x[t] + ϵ x^3[t] == 0
```

pro malé hodnoty ϵ, předpokládáme, že frekvence se změní příspěvkem nelineárního členu, a že může být vyjádřena jako poruchová řada. Nechť Ω je neznámá frekvence periodického řešení strunové rovnice, pak provedeme substituci $\tau = \Omega t$ tak, aby řešení mělo periodu 2π vzhledem k nové proměnné τ.

```
x[τ + 2 π] = x[τ]
```

Takže nelineární strunová rovnice přejde od tvaru

```
Ω^2 x''[τ] + ω^2 x[τ] + ϵ x^3[t] == 0
```

Lindstedtova metoda předpokládá, že

```
x (τ) = x0 (τ) + ϵ x1 (τ) + ...
```

a

```
Ω = Ω0 + ϵ Ω1 + ...
```

Pro $\epsilon = 0$, musí zřejmě platit $\Omega_0 = \omega$

Řešený příklad

Nalezněte Lindstedtovo řešení pro tvrdou strunu v prvním řádu ϵ.

```
x''[t] + ω^2 x[t] + ϵ x^3[t] == 0
```

Řešení:

```
Clear["Global`*"]
n = 1;
x[τ] = Sum[xi[τ] ϵ^i, {i, 0, n}]
Ω = Sum[ωi ϵ^i, {i, 0, n}]
```

$x_0[\tau] + \epsilon\, x_1[\tau]$

$\omega_0 + \epsilon\, \omega_1$

Vložíme rovnici pro tvrdou strunu a řešení rozvedeme.

```
eq1 = Expand[Ω^2 D[x[τ], {τ, 2}] + ω0^2 x[τ] + ϵ (x[τ])^3]
```

$$\omega_0^2 x_0[\tau] + \epsilon\, x_0[\tau]^3 + \epsilon\, \omega_0^2 x_1[\tau] + 3\,\epsilon^2 x_0[\tau]^2 x_1[\tau] + 3\,\epsilon^3 x_0[\tau] x_1[\tau]^2 + \epsilon^4 x_1[\tau]^3 + \omega_0^2 x_0''[\tau] + 2\,\epsilon\,\omega_0\,\omega_1 x_0''[\tau] + \epsilon^2 \omega_1^2 x_0''[\tau] + \epsilon\,\omega_0^2 x_1''[\tau] + 2\,\epsilon^2\,\omega_0\,\omega_1 x_1''[\tau] + \epsilon^3\,\omega_1^2 x_1''[\tau]$$

Seřadíme jednotlivé členy podle řádu v ϵ a vygenerujeme potřebný počet dif. rovnic.

```
eq2 = Collect[eq1, ϵ]
odsystem =
 Table[Expand[CoefficientList[eq2, ϵ][[i]] / ω0^2] == 0, {i, 1, n + 1}] // TableForm
```

$$\omega_0^2 x_0[\tau] + \epsilon^4 x_1[\tau]^3 + \omega_0^2 x_0''[\tau] + \epsilon\left(x_0[\tau]^3 + \omega_0^2 x_1[\tau] + 2\,\omega_0\,\omega_1 x_0''[\tau] + \omega_0^2 x_1''[\tau]\right) + \epsilon^2\left(3\, x_0[\tau]^2 x_1[\tau] + \omega_1^2 x_0''[\tau] + 2\,\omega_0\,\omega_1 x_1''[\tau]\right) + \epsilon^3\left(3\, x_0[\tau] x_1[\tau]^2 + \omega_1^2 x_1''[\tau]\right)$$

$$x_0[\tau] + x_0''[\tau] == 0$$

$$\frac{x_0[\tau]^3}{\omega_0^2} + x_1[\tau] + \frac{2\,\omega_1\, x_0''[\tau]}{\omega_0} + x_1''[\tau] == 0$$

Nyní jsi tyto ODE separátně zobrazíme

```
od0 = odsystem[[1, 1]]
od1 = odsystem[[1, 2]]
```

$$x_0[\tau] + x_0''[\tau] == 0$$

$$\frac{x_0[\tau]^3}{\omega_0^2} + x_1[\tau] + \frac{2\,\omega_1\, x_0''[\tau]}{\omega_0} + x_1''[\tau] == 0$$

První rovnici od0 analyticky vyřešíme za určitých okrajových podmínek, např.

```
Sol0 = DSolve[{od0, x0[0] == A, x0'[0] == 0}, x0[τ], τ]
x0 = x0[τ] /. Sol0[[1]]
```

$\{\{x_0[\tau] \to A \cos[\tau]\}\}$

$A \cos[\tau]$

Tento nulový řád řešení je následně vložen do rovnice *od*1 jako funkce:

```
od1 = od1 /. x0 → Function[τ, A Cos[τ]]
```

$$\frac{A^3\,\mathrm{Cos}[\tau]^3}{\omega_0^2} - \frac{2\,A\,\mathrm{Cos}[\tau]\,\omega_1}{\omega_0} + x_1[\tau] + x_1''[\tau] == 0$$

Poté je i *od*1 vyřešeno za použitím nulových okrajových podmínek, neboť informaci o okrajových podmínkách systému již obsahuje nultý řád řešení.

```
Sol1 = FullSimplify[DSolve[{od1, x1[0] == 0, x1'[0] == 0}, x1[τ], τ]]
```

$$\left\{\left\{x_1[\tau] \to -\frac{A\,\mathrm{Sin}[\tau]\,\left(A^2\,(6\,\tau + \mathrm{Sin}[2\,\tau]) - 16\,\tau\,\omega_0\,\omega_1\right)}{16\,\omega_0^2}\right\}\right\}$$

Členy obsahující τ jsou seskupeny

```
x1 = Collect[x1[τ] /. Sol1[[1]], τ]
```

$$-\frac{A^3\,\mathrm{Sin}[\tau]\,\mathrm{Sin}[2\,\tau]}{16\,\omega_0^2} - \frac{A\,\tau\,\mathrm{Sin}[\tau]\,\left(6\,A^2 - 16\,\omega_0\,\omega_1\right)}{16\,\omega_0^2}$$

Koeficient stojící před τ je položen nule čímž nalézáme první řád opravy k frekvenci.

```
freq = Solve[Coefficient[x1, τ] == 0, ω1]
```

$$\left\{\left\{\omega_1 \to \frac{3\,A^2}{8\,\omega_0}\right\}\right\}$$

Frekvence je do prvního řádu v ϵ dána vztahem:

```
Ω = Ω /. freq[[1]]
```

$$\frac{3\,A^2\,\epsilon}{8\,\omega_0} + \omega_0$$

K vyjádření prvního řádu řešení můžeme například položit $\tau \sin\tau \to 0$ a zjednodušit výraz pomocí příkazu `TrigReduce[]`, nebo příkazem `Simplify[]`:

```
x1 = TrigReduce[x1 /. τ Sin[τ] → 0]
x1p = Simplify[x1 /. τ Sin[τ] → 0]
```

$$\frac{-A^3\,\mathrm{Cos}[\tau] + A^3\,\mathrm{Cos}[3\,\tau]}{32\,\omega_0^2} - \frac{A^3\,\mathrm{Cos}[\tau]\,\mathrm{Sin}[\tau]^2}{8\,\omega_0^2}$$

Celkové řešení do patřičného řádu je:

```
x = x0 + ϵ Collect[x1, A^3]
xp = x0 + ϵ Collect[x1p, A^3]
```

$$A\,\mathrm{Cos}[\tau] + \frac{A^3\,\epsilon\,(-\mathrm{Cos}[\tau] + \mathrm{Cos}[3\,\tau])}{32\,\omega_0^2}$$

$$A\,\mathrm{Cos}[\tau] - \frac{A^3\,\epsilon\,\mathrm{Cos}[\tau]\,\mathrm{Sin}[\tau]^2}{8\,\omega_0^2}$$

Úkoly

1. Analyticky (pomocí programu Mathematica) dokažte následující trigonometrickou identitu:

   ```
   Cos[5 τ] == 16 (Cos[τ])^5 - 20 (Cos[τ])^3 + 5 Cos[τ]
   ```

2. Nalezněte řešení tvrdé struny do třetího řádu a ten porovnejte s analytickým řešení. Diskutujte, proč metoda selhává!

Souhrn kapitoly

V této kapitole jsme představili některé aproximační metody řešení diferenciálních rovnic a ukázali si jejich použití při řešení fyzikálních problému v programu `Wolfram Mathematica`.

Otázky

1. Kdy můžeme použít aproximativní metody? Jaký typ diferenciálních rovnic tak můžeme řešit?
2. Pro jaké případy se používá Lindstedtova metoda?
3. Jak se k řešení používá Taylorova řada?

Kapitola 9

Profil spektrální čáry od rotujícího prstence hmoty

Vladimír Karas

Rychlý náhled: Tato kapitola popisuje vznik spektrálních čar vznikajících v prstenci hmoty rotujícím kolem kompaktního tělesa. Tato kapitola je pouze zadání úlohy, kterou student může vypracovat v rámci závěrečné seminární práce.

Klíčová slova: Spektrální čáry, kompaktní objekty, numerická integrace

Cíle studia: Cílem kapitoly je představit studentům princip vzniku spektrálních čar z rotujícího prstence v okolí kompaktního objektu tak, aby byli studenti schopni samostatně navrhnout a implementovat řešení.

Spektrální profil emisní čáry vznikající v rotujícím prstenci

Spektrální čáry jsou zpravidla vyzařovány určitým zářivým přechodem v atomech zahřátého plynu. Vlnová délka λ (nebo ekvivalentně energie resp. frekvence vyzařovaných fotonů ν) je přesně dána. Následkem pohybu jednotlivých atomů a také vlivem gravitačního pole v místě zdroje se však může výsledná pozorovaná spektrální čára jevit vzdálenému pozorovateli rozšířená, s určitým deformovaným profilem. Také může dojít k posunu její vlnové délky (či energie $E = h\nu$) vlivem Dopplerova jevu a gravitačního červeného posunu.

Pozorovaný zářivý tok v laboratorní soustavě spojené s teleskopem lze symbolicky zapsat jako integraci

$$F(\nu) = \int_{\text{(Over observer's plane)}} I(\nu)_{|\theta=\theta_{\rm obs}}\, \mathrm{d}\mathcal{S}. \tag{9.1}$$

Příklad

Převeďte výše uvedený obecný výraz do vhodného tvaru sumy příspěvků od jednotlivých elementů zdroje záření, který pro účely tohoto cvičení považujte za kruhový prstenec o poloměru R. Numerickou integrací přes azimutální úhel podél prstence určete pozorovaný spektrální profil $F(\nu)$ od spektrální čáry emitované na jednotkové energii prstencem s danou konstantní rotační rychlosti $v(R)$ v ekvatoriální rovině. Objekt je pozorován vzdáleným pozorovatelem pod daným úhlem (tzv. inklinací) v intervalu 0 (podél osy prstence) až 90 (v rovině prstence) stupňů.

Pozn.: světelné paprsky považujeme z přímky. Neuvažujeme na tomto místě zakřivení paprsků v silné gravitaci vlivem efektů obecné teorie relativity. Poloha středu spektrální čáry (tzv. centroidu) zjevně nezávisí na inklinaci disku. Při úplném, self-konzistentním relativistickém řešení tohoto problému není možné oddělovat gravitační posuv od Dopplerova. Vlnová délka centroidu bude na inklinaci záviset, uvážíme-li úplný vztah posuv vlnové délky, anizotropní emisivitu prostředí, závislou na úhlu vyzařování, i gravitační fokusaci světelných paprsků.

Řešení: Vztah mezi pozorovatelem (obs) zaznamenanou a zdrojem emitovanou (em) vlnovou délkou lze zapsat v přibližném tvaru (Gerbal & Pelat 1981; Karas et al. 1995)

$$\lambda_{\rm obs} = \lambda_{\rm em}\,\frac{(1-\beta y)}{\sqrt{1-\beta^2}} \approx \lambda_{\rm em}\left(1+\frac{1}{2}\beta^2-\beta y\right), \tag{9.2}$$

kde $\beta = v/c$, $y = \sin\phi\sin\theta_{\rm obs}$ (ϕ je azimutální úhel v rovině prstence a $\theta_{\rm obs}$ je inklinace pozorovatele, c značí rychlost světla).

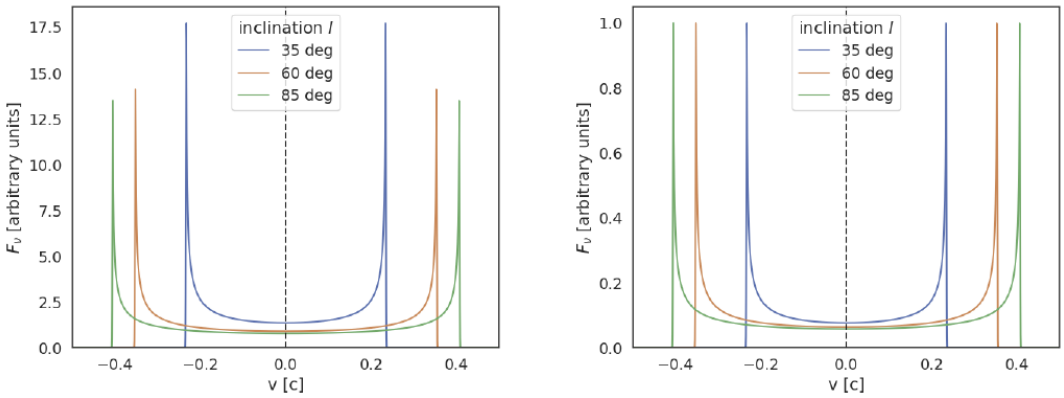
Obdobně lze odvodit vztah pro gravitační červený posuv

$$\lambda_{\rm obs} = \frac{\lambda_{\rm em}}{1-\beta^2}. \tag{9.3}$$

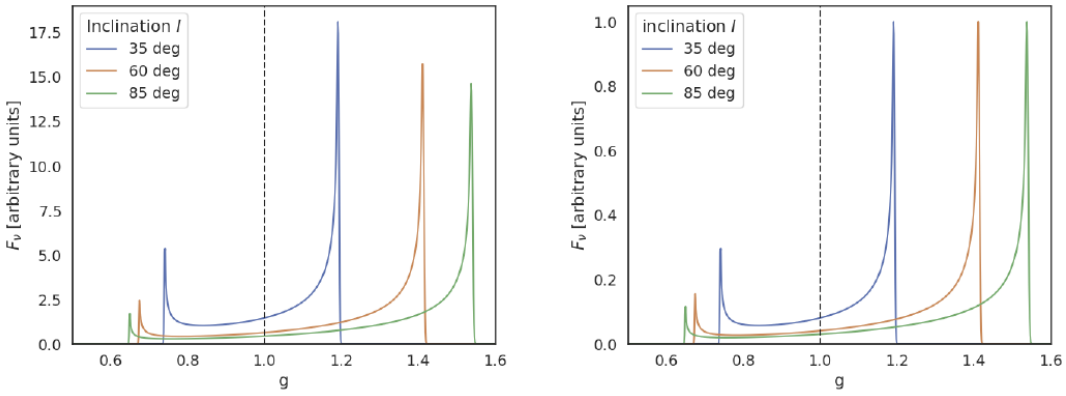
Komentář: Záření plynu, obíhajícího u vnitřního okraje disku, podléhá silnějšímu gravitačnímu a příčnému Dopplerovu jevu, než záření z odlehlejších oblastí. Výsledkem je nesouměrnost pozorovaných čar, které jsou zároveň celkově posunuty k červenému konci spektra. K tomu přistupuje dopplerovsky zesílené záření přibližujícího se materiálu, jež zvýrazňuje vrchol na modré straně čáry. Graf 9.1 ilustruje spektrální profil, kde elementy prstence vydávají záření o frekvenci $\nu = \nu_0$, představující v lokální klidové soustavě úzkou spektrální čáru s charakterem Diracovy funkce δ. Profil se potom vytváří překrytím jednotlivých příspěvků ovlivněných Dopplerovým jevem a gravitačním červeným posuvem. Pozorovaný profil závisí na pozorovatelově inklinaci $\theta_{\rm obs}$.

Úkoly

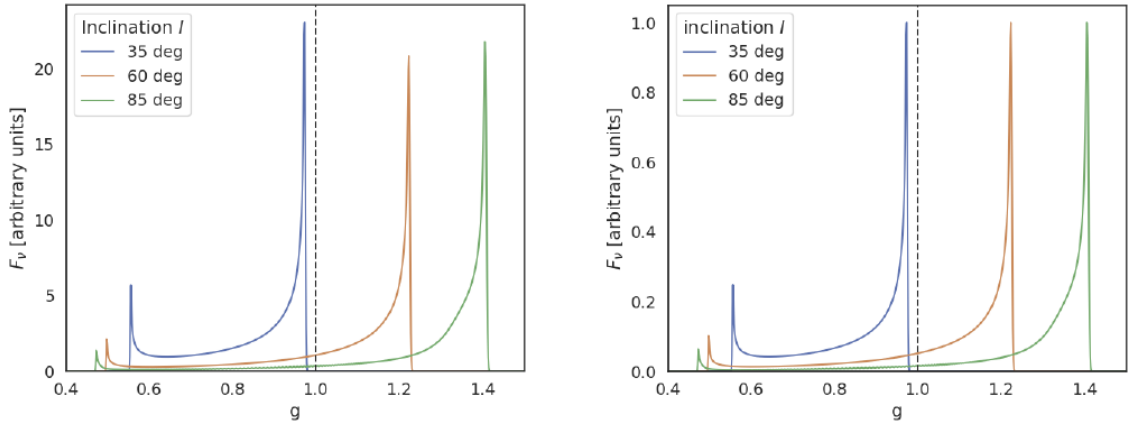
1. Diskutujte význam největší a nejnižší frekvence pozorované čáry. Za předpokladu keplerovské rotace kolem centrálního tělesa o hmotnosti M splňuje orbitální rychlost látky, odpovídající vrcholům v profilech čar, vztah
$$\frac{v_{K\alpha|R=R_{\rm in}}}{v_{K\alpha|R=R_{\rm out}}} = \sqrt{\frac{R_{\rm out}}{R_{\rm in}}}$$
nezávisle na úhlu inklinace. Tento poměr poskytuje první odhad rozměru disku.
2. Integrací přes R zjistěte spektrální profil čáry od radiálně rozlehlého disku. V případě disku se profil čáry získá jeho rozdělením do jednotlivých prstenců o radiální šířce dr, z nichž každý vyzařuje se svou vlastní lokální frekvencí a intenzitou, a následným složením výsledného signálu. Integrace probíhá od určitého vnitřního okraje $R_{\rm in}$ do vnějšího poloměru $R_{\rm out}$. Prostudujte polohu centroidu energie E_c spektální čáry rozlehlého disku.



Obrázek 9.1: Spektrální čáry plynného prstence na poloměru $6\,R_g$ v Newtonovské aproximaci normalizovaný: vlevo – tak, že $\int F_\nu \mathrm{d}\nu = 1$, vpravo – k velikosti peaku modrého posunu na dané spektrální čáře. Černá tečkovaná čára ukazuje vlastní frekvenci.



Obrázek 9.2: Spektrální čáry plynného prstence na poloměru $6\,R_g$ ve speciální relativitě, normalizovaný: vlevo – tak, že $\int F_\nu \mathrm{d}g = 1$, vpravo – k velikosti peaku modrého posunu na dané spektrální čáře. Černá tečkovaná čára ukazuje vlastní frekvenci.



Obrázek 9.3: Spektrální čáry plynného prstence na poloměru $6\,R_g$ ve obecné relativitě, normalizovaný: vlevo – tak, že $\int F_\nu \mathrm{d}g = 1$, vpravo – k velikosti peaku modrého posunu na dané spektrální čáře. Černá tečkovaná čára ukazuje vlastní frekvenci.

Další informace

1 Gerbal, D., & Pelat, D. 1981, "Profile of a line emitted by an accretion disk. Influence of the geometry upon its shape parameters", Astronomy & Astrophysics 95, 18;

2 Karas, V., Lanza, A., & Vokrouhlický, D. 1995, "Emission-line profiles from self-gravitating thin disks", The Astrophysical Journal 440, 108;

3 Kojima, Y. 1991, "The effects of black hole rotation on line profiles from accretion discs", Monthly Notices of the Royal Astronomical Society 250, 629

4 Laor, A. 1991, "Line profiles from a disk around a rotating black hole", The Astrophysical Journal 376, 90

5 Stolc, M. 2019, "Accretion discs in the context of tidal disruption of stars in nuclei of galaxies", Master Thesis, Prague

Souhrn kapitoly

V této kapitole jsme uvedli studenty do problematiky spektrálních čar, které jsou základem pro samostatnou práci.

Otázky

1. Co je zdrojem záření vytvářející spektrální čáry?
2. Jak se mění tvar čar, používáme-li stále přesnější teorii gravitace? Čím je to způsobeno?

Literatura

[1] Stewart, William J. Probability, Markov Chains, Queues and Simulation: The Mathematical Basis of Performance Modeling. [s.l.]: Princeton University Press, 2011. Dostupné online. ISBN 978-1-4008-3281-1

[2] GRINSTEAD, Charles M.; SNELL, J. Laurie. Grinstead & Snell, Úvod do pravděpodobnosti. [s.l.]: Orange Grove Texts, 2009. ISBN 161610046X.

[3] HUBBLE, Edwin. A Relation between Distance and Radial Velocity among Extra-Galactic Nebulae. *Proceedings of the National Academy of Sciences of the United States of America*. 1929, 15, 168-173.

[4] Kapteyn Astronomical Institute. Least squares fitting with kmpfit. Dostupné z https://www.astro.rug.nl/software/kapteyn/kmpfittutorial.html

[5] KIRSHNER, Robert P. Hubble's diagram and cosmic expansion. *Proceedings of the National Academy of Science*. 2004, 101, 8-13.

[6] PRESS William H., TEUKOLSKY Saul A., VETTERLING William T., FLANNERY Brian P. Numerical recipes. The art of scientific computing. Third edition. *Cambridge*. 2007, ISBN-13 978-0-521-88068-8

[7] ČERMÁK, Libor a Rudolf HLAVIČKA. Numerické metody I. In: VUT, FSI, Ústav matematiky [online]. Brno, 1.11.2016. Dostupné z: http://mathonline.fme.vutbr.cz/Numericke-metody-I/sc-8-sr-1-a-11/default.aspx