Univerzita Hradec Králové

Přírodovědecká fakulta

Bakalářská práce

2022 **Jiří Kánský**

UNIVERZITA HRADEC KRÁLOVÉ

Přírodovědecká fakulta

Sbírka úloh z optimalizace využitím knihovny Sympy

Bakalářská práce

Autor: Jiří Kánský

Studijní obor: Finanční a pojistná matematika

Vedoucí práce: doc. Mgr. Bednařík Dušan, Ph.D.

Hradec Králové 2022

Univerzita Hradec Králové
Přírodovědecká fakulta

Zadání bakalářské práce

Autor: **Jiří Kánský**

Studium: S18AM010BP

Studijní program: B1103 Aplikovaná matematika

Studijní obor: Finanční a pojistná matematika

Název bakalářské práce: **Sbírka úloh z optimalizace využitím knihovny Sympy**

Název bakalářské práce AJ: Collection of exercises from optimization theory solving with the Python Sympy library

Cíl, metody, literatura, předpoklady:

Cílem práce je prezentovat některé možnosti knihovny Sympy při řešení úloh z teorie optimalizace. Při použití této knihovny se používá syntaxe programovacího jazyka Python. Sbírka bude ukazovat
i prezentační možnosti knihoven pythonu pro zobrazení dat a matematických formulí.

1. Mark Pilgrim: Ponořme se do Pyton(u) 3, CZ.NIC (2010)

2. Sympy, dokumentace na webu (https://www.sympy.org/cs/)

Garantující pracoviště: Katedra matematiky,
Přírodovědecká fakulta

Vedoucí práce: doc. Mgr. Dušan Bednařík, Ph.D.

Oponent: Mgr. Tomáš Zuščák, Ph.D.

Datum zadání závěrečné práce: 11.9.2020

Prohlášení:

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a že jsem v seznamu použité literatury uvedl všechny prameny, z kterých jsem vycházel.

V Hradci Králové Jiří Kánský

Anotace:

KÁNSKÝ, J. *Sbírka úloh z optimalizace využitím knihovny Sympy*. Hradec Králové, 2022. Bakalářská práce na Přírodovědecké fakultě Univerzity Hradec Králové. Vedoucí bakalářské práce Dušan Bednařík. 97 s.

Ve své bakalářské práci prezentuji možnosti knihovny Sympy při řešení úloh z teorie optimalizace. V úvodní části se zabývám programovacím jazykem Python a jeho historií. Dále popisuji knihovny jazyka Python, konkrétněji se zabývám knihovnou Sympy. Na závěr vytvářím algoritmy na výpočet extrémů funkcí jedné a dvou proměnných v rámci knihovny Sympy pomocí programovacího jazyka Python.

Klíčová slova:

optimalizace, Python, Sympy, kód

Annotation:

KÁNSKÝ, J. *Collection of exercises from optimization theory solving with the Python Sympy library*. Hradec Králové, 2022. Bachelor Thesis at Faculty of Science University of Hradec Králové. Thesis supervisor Dušan Bednařík. 97 p.

In my bachelor thesis I present possibilities of Python library Sympy when dealing with problems from optimization theory. In the introductory section I am concerned with programming language Python a its history. The thesis continues in description of Python libraries, specifically library Sympy. Lastly, I create algorithms for finding extrama of functions of one or two real variables via Sympy library through programming language Python.

Keywords:

optimization, Python, Sympy, code

Poděkování:

Touto cestou bych chtěl poděkovat svému vedoucímu mé bakalářské práce doc. Mgr. Dušanu Bednaříkovi, Ph.D. za cenné rady, které mi napomohly při tvorbě této bakalářské práce.

Obsah

Úvod

V této práci prezentuji některé z možností knihovny Sympy při řešení úloh z teorie optimalizace. Při použití této knihovny se používá syntaxe programovacího jazyka Python.

V první kapitole charakterizuji programovací jazyk Python. Nastíním jeho historii od vzniku v roce 1989 až do současnosti, kdy je volně dostupná verze Python 3.10.4. Dále popišu nástroje řídící struktury programovacího jazyku, které čtenář potřebuje znát, aby rozumněl praktickému zpracování práce. Patří mezi ně datové typy, početní operátory a další nástroje řídící struktury.

V druhé kapitole vysvětlím pojem knihovny v rámci programovacího jazyka Python a charakterizuji knihovnu Sympy, kterou jsem využíval při tvorbě funkcí na řešení optimalizačních problémů z matematického hlediska. Funkce, které jsem využil v praktické části práce, jsou zde stručně vysvětleny a popsány. Nadále také lehce nastíním ostatní knihovny a jejich funkce, se kterými jsem pracoval.

Ve třetí kapitole prezentuji funkce vlastní výroby, které jsem vytvořil za účelem řešení optimalizačních problémů. Tyto funkce byly zpracovány v rámci interpreta programovacího jazyka Python a využívájí převážně knihovny Sympy v jejich struktuře. Jsou to příklady typu hledání extrémů funkcí jedné nebo dvou reálných proměnných. Kromě obecně využívaných postupů při řešení problémů tohoto typu jsem také vytvořil funkce reprezentující gradientní sestup a Newtonovu metodu.

Cílem práce je tedy ukázat možnosti řešení optimalizačních úloh v prostoru pro symbolickou matematiku, kterou poskytuje knihovna Sympy.

1. Python

1.1 Charakteristika programovacího jazyka Python

Python je skriptovací programovací jazyk vysoké úrovně, který umožňuje vytvářet přesné a logické aplikace na malé i velké projekty. Toto je možné zejména proto, že strukturu Pythonu tvoří silné konstrukty a pravidelné užívání objektů a objektově orientovaného programování. Důležitou součástí Pythonu jsou datové typy (stringy, seznamy, slovníky apod.), běžné příkazy pro kontrolu (if, if-else, if-elif-else), a iterátory (for, while). Python tvoří několik úrovní organizační struktury, jako jsou funkce, třídy, moduly a balíčky. Tyto části napomáhají organizaci kódu. Velice užitečným a obsáhlým příkladem je standartní knihovna Pythonu. (Kuhlman, 2009)

Python se lze jednoduše naučit. Jeho elegantní syntax a dynamické psaní spolu s interpretovanou povahou z něj dělá ideální programovací jazyklehce použitelný pro rapidní tvorbu aplikací v mnoha oborech na většině platformách.

Python a jeho rozsáhlou standartní knihovnu najdete volně dostupné ve zdrojové nebo binární formě z webové stránky https://www.python.org/ a může být volně distribuován. Jedná se o open source projekt, který si může pořídit skoro každý, a to zcela zdarma. (Python docs, 2022)

1.2 Historie

Autorem programovacího jazyka Python je Guido van Rossum.

Guido van Rossum získal rozsáhlé zkušenosti s implementací programovacích jazyků při práci na programovacím jazyce ABC v CWI. Původ mnoha rysů Pythonu leží právě v ABC, včetně využití odrážek a zahrnutí datových typů vysoké úrovně. Detaily se však v Pythonu liší.

Rossum měl hodně problémů s ABC, ale zároveň se mu líbila spousta jeho vlastností. Problém byl, že nebylo možné ABC rozšířit natolik, aby se tyto nedostatky upravily – jeden z těchto nedostatků byla právě jeho nerozšiřitelnost.

Když Rossum pracoval v CWI na operačním systému Amoeba, tak spolu s kolegy usoudil, že na administraci systému potřebují něco nového, a programy v C nebo Bourne shell skripty na to nestačily. Amoeba měla vlastní systémové rozhraní, do kterého nebylo snadné se dostat zkrze Bourne shell.

Zjistil, že skriptovací jazyk jako ABC s přístupem k systémovým výzvám Amoeby bylo právě to, co potřebovali. Rozhodl se, že vytvoří jazyk, který není exkluzivní pouze pro Amoebu, ale je volně rozšiřitelný i na jiné platformy.

Na konci roku 1989 na tom začal pracovat a v roce 1990 byl Python využitý v projektu Amoeba s velkým úspěchem, zpětná vazba od kolegů umožnila mnoho úprav. (Venners, 2003; Python docs, 2022)

V únoru roku 1991 Rossum zveřejnil Python 0.9.0 na alt.sources.

V roce 1994 vzniklo primární diskuzní fórum comp.lang.python, důkaz o rostoucím počtu uživatelů Pythonu.

Python dosáhnul verze 1.0 v lednu roku 1994.

Poslední verze, která vyšla, když byl ještě Rossum součástí ICW, byla 1.2. V roce 1995 na své práci pokračoval v CNRI v Restonu ve Virginii, kde vydal několik verzí.

Během jeho působení v CNRI Rossum zahájil Computer Programming for Everybody (CP4E) inciativ. Chtěl, aby programování bylo dostupné pro větší skupinu lidí. Python zde hrál centrální roli převážně proto, že se soustředí na čistý syntax. (Python docs, 2022; Rossum, 1999)

V roce 2000 se tým developerů přesunul na BeOpen.com za účelem vytvořit BeOpen PythonLabs tým pod vedením Domenica Merendy. CNRI požádala o vydání verze 1.6, s tím, že verze bude obsahovat vše, co doposud tým vytvořil při práci na Pythonu.

Python 2.0 byla jediná verze vydaná pod BeOpen. Po odstoupení od BeOpen, Guido van Rossum a ostatní PythonLab developeři se přidali k Digital Creations.

Verze 2.0 byla vydána v říjnu 2000, odtud se Python 2.x postupně vyvíjel.

Python 2.1 byl podobný pythonu 1.6.1, a zároveň verzi 2.0. Lincence byla přejmenována na Python Software Foundation License. Kód, dokumentace a specifikace, které byly přidány od alpha vydání verze 2.1, patří Python Software Foundation (PSF), neziskové organizaci vzniklé v roce 2001. (Python docs, 2022)

V letech 2001 až 2009 byly postupně vydávány nové verze Pythonu 2.2 až 2.7.

V listopadu 2014 bylo oznámeno, že verze Pythonu 2.7 bude podporována do roku 2020, ale uživatelé byli povzbuzeni se přesunout na Python 3 co nejdříve.

Podpora Python 2.7 skončila 1. ledna 2020, kdy se zastavil rozvoj. Poslední verze 2.7.18 byla vydána 20. dubna 2020, a označovala konec Python 2.

Python 3.0 byl vydán 8. prosince 2008 a získává aktualizace dodnes.

Nejnovější verze 3.10.4 byla vydána 4. října 2021.

(Python docs, 2022)

1.2 Datové typy

Při programování pracujeme s různými typy dat. Tyto typy jsou běžně definovány oborem hodnot a operacemi, které s nimi lze provádět.

Součástí programovacího jazyka je definice základních datových typů, z těchto typů pak lze nadále vytvářet nové typy složené.

V Pythonu každá hodnota nabývá určitého datového typu, ale nemusíme nutně deklarovat datový typ jakékoliv proměnné. To protože když přiřadíme proměnné hodnotu, tak Python sám vyhodnotí a zjistí, o jaký datový typ se vlastně jedná.

Hlavní přirozené datové typy Pythonu:

- Boolean – nabývá hodnot True nebo False
- Čísla – celá čísla (**int**; 1 a 2), reálná (**float**; 1.1 a 2.1), zlomky (**fraction**; 1/2 a 2/3), nebo čísla komplexní
- Řetězce – posloupnosti Unicode znaků,
- Bajty a pole bajtů – například obrázky,
- Seznam – uspořádané posloupnosti hodnot,
- N-tice – uspořádané, neměnné posloupnosti hodnot,
- Množiny – neuspořádané kolekce hodnot
- Slovníky – neuspořádané kolekce dvojic klíč-hodnota.

Protože je v Pythonu všechno objekt, tak typů je samozřejmě mnohem více, třeba jako modul, třída, funkce atd. (Pilgrim, 2010; Python docs, 2022)

Jelikož zde pracujeme s knihovnou Sympy, tak nás převážně budou zajímat pouze číselné operátory a seznamy.

1.2.1 Operátory

Pythonu lze využít jako obyčejné kalkulačky. Stačí pouze do interpretora zadat rovnici a vypíše nám výsledek. Syntax je přímočarý, tj. operátory +,-,*,/ fungují stejně jako ve většině ostatních jazyků. Například takto:

```
>>> 3 + 7
10
>>>100-2*15
70
>>> 8/5
1.6
```

Celá čísla mají datový typ **int**, zatímco ta s desetinou čárkou **float**.

Operátor podílu (/) vždy vrací float. Aby nám Python vrátil celé číslo, můžeme využít operator (//), který vynechá vše za desetinnou čárkou. Pro získání zbytku využijeme operator (%).

```
>>> 17 / 3
5.666666666666667
>>> 17 // 3
5
>>> 17 % 3
2
```

V Pythonu se na umocnění využívá operátor (**).

```
>>>3**2
9
>>>2**5
32
```

Operátor rovná se (=) se využívá při deklaraci proměnných. Po deklaraci se však nic nezobrazí před další výzvou.

```
>>> zakladna = 5
>>> vyska = 3*4
>>> zakladna*vyska/2
15
```

V případě, že proměnná není definována, pak po zavolání nám Python vrátí NameError.

```
>>> f
NameError: name 'f' is not defined
```

(Pilgrim, 2010; Python docs, 2022)

1.2.2 Seznamy

Seznam je nejuniverzálnější ze složených datových typů. Lze ho zapsat jako seznam hodnot oddělených čárkou uvnitř hranatých závorek. Seznam může obsahovat položky různých datových typu, obvykle ale obsahuje položky typu stejného.

```
>>>seznam=[1, 2, 3, 4, 5]
>>>seznam
[1, 2, 3, 4, 5]
```

Se seznamem lze pracovat. Například můžeme vybírat specifické prvky nebo ho lze oříznout.

```
>>>seznam [0]
1
>>>seznam [-1]
5
>>>seznam [-3:]
[3, 4, 5]
```

Seznamy lze prodloužit.

```
>>>seznam+[6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Můžeme měnit jejich položky.

```
>>>seznam[2]=5
>>>seznam
[1, 2, 5, 4, 5]
```

Nebo jednotlivé položky přidávat.

```
 >>>seznam.append(6)
>>>seznam
[1, 2, 5, 4, 5, 6]
```

Můžeme také využít příkaz len(), který spočítá počet položek uvnitř seznamu.

```
>>>seznam=[1, 2, 3, 4, 5]
>>>len(seznam)
5
```

Lze také vytvořit seznam dvou seznamů.

```
>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> x
[['a', 'b', 'c'], [1, 2, 3]]
```

(Pilgrim, 2010; Python docs, 2022)

Toto byly nejčastější a nejobecnější možnosti jak s jednotlivými seznamy nakládat.

1.3 Nástroje řídící struktury

Python využívá nástroje řídící struktury podobně jako ostatní programovací jazyky, s malou změnou.

1.3.1 While

Cyklus **while** provádí zadaný příkaz, dokud daná podmínka stále platí.

Zde pomocí **while** tvoříme Fibonacciho posloupnost.

```
>>>a, b=0, 1
>>>while a<100:
. . . . . .     print(a, end = “,”)
. . . . . .     a, b = b, a+b
. . . . . .
0,1,1,2,3,5,8,13,21,34,55,89
```

(Pilgrim, 2010; Python docs, 2022)

1.3.2 If

Jeden z nejznámějších příkazů je příkaz **if**. Například:

```
>>> x = int(input(“Zadejte celé číslo: ”)
Zadejte celé číslo: -3
>>> if x > 0:
. . . . . .     print(“Číslo je kladné”)
>>> elif x == 0:
. . . . . .     print(“Číslo je nula”)
>>> else:
. . . . . .     print(“Číslo je záporné”)
Číslo je záporné
```

Částí elif může být několik nebo nemusí být žádné. Část else je nepovinná. Příkazové slovo “elif” je zkráceně “else if” a užívá se pro vyhnutí se nadměrnému odsazování.

(Pilgrim, 2010; Python docs, 2022)

1.3.3 For

V Pythonu **for** iteruje položky posloupnosti (seznamu nebo stringu) v pořadí v jakém jsou uspořádány. Například:

```
>>> slova = [“paranoia”, “tyrannosaurus”, “kocka”]
>>> for a in slova:
. . . . . .       print(a, len(a))
paranoia 8
tyrannosaurus 13
kocka 5
```

(Pilgrim, 2010; Python docs, 2022)

1.3.4 Break, continue a else v cyklu

Příkaz **break** se využívá v nejvnitřnější části cyklu, kde ho přeruší.

Cykly mohou mít klauzuli else. Ta proběhne, když se cyklus přeruší, protože položky byly využity (pro cyklus **for**) nebo protože podmínka již není splněna (cyklus **while**), ale ne když je cyklus přerušen příkazem **break**. Například:

```
>>> for n in range(2, 10):
. . . . . .       for x in range (2, n):
. . . . . .             if n % x == 0:
. . . . . .                   print(n, “se rovná”, x, “*”, n//x)
. . . . . .                   break
. . . . . .       else:
. . . . . .             print(n, “je prvočíslo”)
2 je prvočíslo
3 je prvočíslo
4 se rovná 2*2
5 je prvočíslo
6 se rovná 2*3
7 je prvočíslo
8 se rovná 2*4
9 se rovná 3*3
```

Příkaz **continue** pokračuje další iteraci cyklu.

```
>>> for n in range(2, 10):
. . . . . .       if n % 2 == 0:
. . . . . . . . . . . .       print(n, " je sudé číslo")
. . . . . . . . . . . .       continue
. . . . . .       print(n, " je liché číslo")
2 je sudé číslo
3 je liché číslo
4 je sudé číslo
5 je liché číslo
6 je sudé číslo
7 je liché číslo
8 je sudé číslo
9 je liché číslo
```

(Pilgrim, 2010; Python docs, 2022)

1.3.5 Pass

Příkaz **pass** se používá, pokud s hodnotou proměnné nechceme nadále pracovat. Například:

```
>>> while True:
. . . . .       pass
```

(Python docs, 2022)

1.4 Definování funkcí

Můžeme vytvořit funkci, která vypíše Fibonacciho posloupnost do libovolného čísla.

```
>>> def fibo(n):
. . . . . """Vypiš Fibonacciho posloupnost v rámci n """
. . . . . a, b=0, 1
. . . . . while a<n:
. . . . . . . . . . . . print(a, end="\" \"" )
. . . . . . . . . . . . a, b = b, a+b
. . . . . print()
>>> fibo(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

Příkaz **def** představí definici funkce. Musí být následován jménem funkce a v závorce za jménem musí být seznam parametrů. Příkazy, které tvoří tělo funkce, začínají na další lince a musí být odstaveny.

První příkaz těla funkce může být doslovný řetězec. Tento řetězec je dokumentačním řetězcem funkce, neboli docstring. Existují nástroje, které používají docstring na automatické produkování online nebo tištěné dokumentace nebo, aby si uživatel mohl kód důkladně prohlédnout.

Provedením funkce vytvoříme novou lokální tabulku symbolů, která je využívána pro lokální proměnné dané funkce. Přesněji, všechny deklarace proměnných uvnitř funkce ukládají svoji hodnotu v lokální tabulce symbolů té funkce.

Tabulka symbolů je datová struktura, do které interpret nebo překladač ukládá všechny identifikátory ve zdrojovém kódu programu.

Při volání na určitou proměnnou se program nejprve podívá na lokální tabulku symbolů, potom na lokální tabulku symbolů vepsaných funkcí, poté na globální tabulku symbolů a nakonec na tabulku vestavěných jmen. To znamená, že globální proměnné a proměnné vepsaných funkcí nemůžeme deklarovat v naší funkci, ale můžeme na ně odkazovat.

Aktuální parametry dané funkce jsou představeny v lokální tabulce symbolů té dané funkce, když tu funkci zavoláme.

Když funkce zvolá jinou funkci nebo zvolá sama sebe, vytvoří tím novou tabulku lokálních symbolů pro tento konkrétní případ.

(Python docs, 2022)

2. Knihovny

Při vypnutí interpreta všechny definice (funkce a proměnné), které jste doposud vytvořili, Python zapomene.

V případě, že píšete poměrně obsáhlý program, je mnohem jednodušší ho rozdělit na několik části. Může stát, že budete potřebovat nějakou funkci z předešlého programu. Je ideální mít po ruce funkci použitelnou ve více programech bez opakovaného kopírování její definice.

Jako podporu pro uživatele v Pythonu existuje cesta, jak dávat jednotlivé definice do souboru. Tento soubor se nazývá modul.

Knihovna Pythonu je znovupoužitelný kus kódu, který je sestaven ze sbírky modulů. Má v sobě předinstalovanou knihovnu s názvem The Python Standard Library.

Standardní knihovna Pythonu je velice rozsáhlá a nabízí ohromnou škálu možností.

Knihovna obsahuje vestavěné moduly psané v jazyce C, které nabízí úroveň funkcionality, která by jinak byla nedostupná programátorům Pythonu, a nabízí standardizovaná řešení k problémům, které se bežně vyskytují v každodenním programování. Některé z těchto modulů jsou explicitně navrhnuty tak, aby podporovaly a zlepšovaly přenosnost pythonovských programů pomocí oddalování se od specifických na více neutrální rozhraní. (Python Docs, 2022)

2.1 Import

Moduly, tedy i knihovny, lze zavolat právě pomocí funkce **_import_()**. Příkaz import tuto funkci zavolá. Import je nejužívanější cesta, jak využívat moduly.

Příkaz využívá dvě operace dohromady. Nejprve vyhledá daný modul a pak přiřadí svůj výsledek jménu v rámci lokálního prostoru. Když je modul poprvé importován, Python ho vyhledá a v případě, že ho najde, vytvoří objekt modul, čímž ho inicializuje.

Hodnota, kterou funkce vrací je vyditelná v outputu počátečního kódu.

```
>>> import math
>>> print(math.pi)
3.141592653589793
```

V kódu výše je vidět, že po importování modulu math, proměnné v rámci tohoto modulu lze zavolat tak, že o modulu math uvažujeme jako o třídě a pi jako o jejím objektu.

Tedy stylem **trida.objekt**.

Hodnotu π nám vrací funkce _import()_.

Objekty modulu lze také importovat jednotlivě pomocí příkazu **from**.

Například:

```
>>> from math import pi
>>> print(pi)
3.141592653589793
```

Naopak všechny funkce a konstanty můžeme importovat pomocí *.

```
>>> from math import*
>>> print(pi)
>>> print(factorial(6))
3.141592653589793
720
```

V případě, že funkce _import()_ nenajde požadovaný modul, vrátí nám ModuleNotFoundError.

(Python docs, 2022; GeekforGeeks 2021)

2.2 Sympy

Sympy je jedna z knihoven Pythonu zaměřená na symbolickou matematiku.

Cílem této knihovny je stát se plnohodnotným počítačovým algebraickým systémem, při zachování co nejjednoduššího kódu, aby byl srozumitelný a snadno rozšiřitelný. Je psán výhradně v Pythonu.

Mezi hlavní způsobilosti Sympy patří základní aritmetika, zjednodušování, rozšiřování, funkce, substituce, čísla, nekomutativní symboly a porovnávání vzorů. Zvládá mnohočleny, matematickou analýzu, rovnice, kombinatoriku, diskrétní matematiku, matice, geometrii, vykreslování grafů, fyziku, statistiku a kryptografii.

Sympy vytváří a udržuje velká skupina přispěvatelů. Celý projekt byl zahájen v roce 2005 Ondřejem Čertíkem a dnes se na něm podílí vice než 500 osob. (Sympy, 2022)

Symbolický počet se vypořádává s výpočtem matematických objektů symbolicky. To znamená, že matematické objekty jsou reprezentovány naprosto přesně, nikoli přibližně, a matematické výrazy s nevyhodnocenými proměnnými zůstávají v symbolické formě.

Pokud bychom chtěli využít Pythonu na spočítání druhé odmocniny nějakého výrazu, udělali bychom to nejspíše takto:

```
>>> import math
>>> math.sqrt(9)
3.0
```

Odmocnina z čísla 9 vrací celé číslo 3, když ale odmocníme jiné číslo, například 8:

```
>>> math.sqrt(8)
2.82842712475
```

Vrátí pouze přibližný výsledek, protože 2.82842712475 není naprosto přesná druhá odmocnina čísla 8. Kdyby nás zajímala pouze desetinná hodnota druhé odmocniny z osmi, tak bychom dál nemuseli nic řešit.

Může se ale stát, že budeme chtít něco jiného, než jenom desetinné číslo. Vzpomeňte si, že $\sqrt{8} = \sqrt{4 * 2} = 2\sqrt{2}$. Z desetinného čísla, které jsme získali výše, bychom toto těžko vyhodnotily.

S využítím knihovny Sympy tento problem nenastane.

V případě, že odmocníme pomocí Sympy, tak druhé odmocniny čísel, které netvoří perfektní čtverce, jsou nevyhodnoceny.

```
>>> import sympy
>>> sympy.sqrt(3)
sqrt(3)
```

Následovně můžete spozorovat, že sympy dokonce i tyto neperfektní odmocniny dokáže zjednodušit.

```
>>> sympy.sqrt(3)
2*sqrt(2)
```

Příklad výše znázorňuje, jak je možné manipulovat s iracionálními čísly pomocí Sympy.

Sympy toho však dokáže mnohem více. Symbolické početní systémy (nebo také počítačové algebraické systémy, zkráceně CAS) jako Sympy jsou schopny počítat se symbolickými výrazy, které obsahují proměnné.

V Sympy se proměnné definují pomocí funkce **symbols()**.

Symbolický výraz, který reprezentuje matematický výraz $x + 2y$ definujeme následovně:

```
>>> from sympy import symbols
>>> x, y = symbols("x y")
>>> vyraz = x + 2*y
>>> vyraz
x + 2*y
```

S výrazem si můžeme nadále pohrávat:

```
>>> vyraz + 3
x + 2*y + 3
>>> vyraz – x
2*y
```

Jak lze zpozorovat výše, v případě vyraz – x nám program nevrátil x+2*y-x, ale už upravené 2*y. Nemusí se to v Sympy stát vždy:

```
>>> x*vyraz
x*(x+2*y)
```

Názorně se výraz neupravil. Toto se v Sympy stává celkem běžně. Kromě nejjednodušších úprav Sympy neprovádí úpravy automaticky. To je proto, že se nám mohou hodit oba tvary. V rámci Sympy existují funkce, které nám uvede stávající formy výrazů:

```
>>> from sympy import expand, factor
>>> rozsireny_vyraz = expand(x*vyraz)
>>> rozsireny_vyraz
x**2 + 2*x*y
>>> factor(rozsireny_vyraz)
x*(x+2*y)
```

(Sympy docs, 2022)

2.2.1 Funkce využité v rámci práce

Následující část představuje funkce v rámci knihovny Sympy, které byly využity při tvorbě praktické části práce.

Symbol():

Symbol je nejdůležitější třída knihovny Sympy. Pro práci v rámci symbolické matematiky potřebujeme symboly. Proměnné v rámci Sympy jsou tedy objekty třídy Symbol.

```
>>> from sympy import Symbol
>>> x = Symbol("x")
>>> y = Symbol("y")
>>> vyraz = x**2 + y**2
>>> vyraz
x² + y²
```

Symbolem můžeme deklarovat i celý řetězec.

```
>>> a = Symbol(“beton”)
>>> a**3
beton^3
```

Příkaz Symbol() vytvoří pouze jednu proměnnou, ale při použití symbols() jich můžeme vytvořit kolik chceme.

```
>>> from sympy import symbols
>>> x, y = symbols(“x y”)
```

V rámci Sympy existuje modul abc, který obsahuje všechna písmena latinské a řecké abecedy. Můžeme tedy použít následující metodu.

```
>>> from sympy.abc import x, y
```

V abc existují následující předdefinované symboly.

```
>>> from sympy.abc import _clash1, _clash2
>>> _clash1
{'C': C, 'O': O, 'Q': Q, 'N': N, 'I': I, 'E': E, 'S': S}
>>> _clash2
{'beta': beta, 'zeta': zeta, 'gamma': gamma, 'pi': pi}
```

Pro písmena, která existují v obou abecedách, využíváme Symbol() objekt.

Můžeme take volat indexované symboly následujícím způsobem.

```
>>> symbols("x:3")
(x_0, x_1, x_2)
```

Na řetězce to funguje také.

```
>>> symbols("zebra(1:5)")
(zebra_1, zebra_2, zebra_3, zebra_4)
```

(Sympy docs, 2022)

diff():

Příkaz diff() provede derivaci.

```
>>> diff(cos(x),x)
-sin(x)
```

Lze také derivovat víckrát za sebou. Za účelem derivace vyššího stupně stačí za derivovanou funkcí opakovat proměnnou tolikrát, kolikrát chceme derivovat, nebo pouze napsat číslo.

```
>>> diff(x**4, x, x, x)
24x
>>> diff(x**4, x, 3)
24x
```

Můžeme také derivovat podle různých proměnných najednou. Stačí je jenom zapsat ve správném pořadí.

```
>>> vyraz = exp(x*y*z)
>>> diff(vyraz, x, y, y, z, z, z, z)
x^3*y^2*(x^3*y^3*z^3 + 14*x^2*y^2*z^2 + 52*x*y*z + 48)*exp(x*y*z)
```

Lze zavolat diff() jako metodu.

```
>>> vyraz.diff(x, y, y, z, 4)
x^3*y^2*(x^3*y^3*z^3 + 14*x^2*y^2*z^2 + 52*x*y*z + 48)*exp(x*y*z)
```

V obou metodách není rozdíl.

(Sympy docs, 2022)

derive_by_array:

Běžnou operaci derivace v rámci Sympy lze rozšířit na derivaci s ohledem na pole, jestliže daná pole obsahují symboly nebo výrazy vhodné pro derivace.

Představme si, že máme dvě pole $A_{i_1,i_2,\cdots,i_N}$ a $X_{j_1,j_2,\cdots,j_N}$. Derivace těchto polí vrací nové pole B.

$$B_{i_1,\cdots,i_N,j_1,\cdots j_N} \coloneqq \frac{\partial A_{i_1,i_2,\cdots,i_N}}{\partial X_{j_1,j_2,\cdots,j_N}}$$

Funkce derive_by_array tuto operaci provádí.

V případě skaláry se chová jako obyčejná derivace.

```
>>> derive_by_array(sin(x*y), x)
y cos(xy)
```

Skalár derivovaný polem vypadá následovně.

```
>>> derive_by_array(sin(x*y), [x, y, z])
[ycos(xy) xcos(xy) 0]
```

Pole derivované polem vypadá takto.

```
>>> derive_by_array([exp(x), sin(y*z), x], [x, y, z])
⎡e^x     0        1⎤
⎢                  ⎥
⎢ 0   zcos(yz)    0⎥
⎢                  ⎥
⎣ 0   ycos(yz)    0⎦
```

(Sympy docs, 2022)

sqrt():

Příkaz sqrt() provádí druhou odmocninu dané funkce.

Metoda sqrt je předvedena celkem zřetelně v úvodním seznámení s knihovnou Sympy.

solve():

Příkaz solve() se využívá na řešení algebraických rovnic. Předpokladem je, že všechny rovnice jsou položeny nule.

Následující rovnici $x^2 = 1$ vyřešíme pomocí solve() takto.

```
>>> solve(x**2 -1, x)
[-1, 1]
```

První argument je vždy funkce položená nule a druhý je symbol, pro který funkci řešíme.

Solve() lze využít při nerovnosti.

```
>>> solve(x < 3)
−∞ < x ∧ x < 3
```

V případě více proměnných může využít parametr dict=True, a zobrazí se nám přesně kolik se jaká proměnná rovná.

```
>>> solve([x - 3, y - 1], dict=True)
[{x:3, y:1}]
```

Jeden výraz s jedním symbolem.

```
>>> solve(x-3, x)
3
>>> solve(x-y, x)
y
```

Jeden výraz bez symbolu.

```
>>> solve(3, x)
[]
>>> solve(x-3, y)
[]
```

V případě, že nezadáme symboly, pak funkce solve() vyřeší rovnici pro všechny symboly.

```
>>> solve(x - 3)
[3]
>>> solve(x**2 - y**2)
[{x: -y}, {x: y}]
>>> solve(z**2*x**2 - z**2*y**2)
[{x: -y}, {x: y}, {z: 0}]
```

Jestliže zvolíme objekt, který není Symbol, tak je izolován algebraicky a můžeme získat implicitní výsledek.

```
>>> solve(f(x) - x, f(x))
[x]
>>> solve(f(x).diff(x) - f(x) - x, f(x).diff(x))
[x + f(x)]
```

(Sympy docs, 2022)

solveset():

Podobně jako solve(), tato funkce vyřeší zadanou funkci.

Rozdíl je v tom, že vrací objekt typu set, kde tento objekt vyřeší ostatní detaily sám. Jestliže se stane, že solveset() nezná všechny řešení, potom vrátí objekt typu Conditionset s částečným řešením.

Vstupní parametry funkce solveset() jsou pouze rovnice, řešené proměnné a dobrovolný argument domain, který definuje interval, na kterém chceme funkci řešit.

Solveset může vracet nekonečně mnoho řešení. Například při řešení rovnice $\sin(x) = 0$ vrací $\{2n\pi \mid n \in \mathbb{Z}\} \cup \{2n\pi + \pi \mid n \in \mathbb{Z}\}$, přitom solve() vrací pouze $[0, \pi]$.

nonlinsolve():

Funkce nonlinsolve() vyřeší systém o N nelineárních rovnicích s M proměnnými. Vrací reálné i komplexní řešení.

Systém je v tomto případě seznam zadaných rovnic a symboly jsou všechny symboly zadané v seznamovém tvaru.

Vrací konečný soubor řešení uspořádaných n-tic symbolů, pro které má systém řešení.

Pořadí řešení je stejné jako pořadí zadání symbolů.

```
>>> nonlinsolve([x*y - 1, 4*x**2 + y**2 - 5], [x, y])
```

$$\left\{(-1,-1),\left(-\frac{1}{2},-2\right),\left(\frac{1}{2},2\right),(1,1)\right\}$$

Zvládne řešit pro soustavu s kladným počtem dimenzí.

```
>>> nonlinsolve([(x+y)**2 - 4, x + y - 2], [x, y])
```

$$\{(2-y,y)\}$$

V případě, že nějaké z rovnic nejsou mnohočleny, nonlinsolve() zavolá funkci substitution a vrací reálná a komplexní řešení, jestliže existují.

```
>>> nonlinsolve([exp(x) - sin(y), y**2 - 4], [x, y])
```

$$\left\{\left(\left\{i(2n\pi+\pi)+log\left(sin(2)\right)\middle|n\in\mathbb{Z}\right\},-2\right),\left(\left\{2ni\pi+log\left(sin(2)\right)\middle|n\in\mathbb{Z}\right\},2\right)\right\}$$

V případě, že soustava je nelineární s nulovou dimenzí, tak vrací obě řešení pomocí funkce solve_poly_system().

```
>>> nonlinsolve([x**2 - 2*y**2 -2, x*y - 2], [x, y])
```

$$\left\{(-2,-1),(2,1),\left(-\sqrt{2}i,\sqrt{2}i\ \right),\left(\sqrt{2}i,-\sqrt{2}i\right)\right\}$$

(Sympy docs, 2022)

subs():

Jedna z nejzákladnějších operací při práci s matematickými výrazy je substituce.

Příkaz subs() tuto operaci provádí záměnou všech instancí první proměnné druhou následovně.

```
>>> vyraz = sin(x)
>>> vyraz.subs(x,y)
sin(y)
```

Substituci běžně provádíme z jednoho ze dvou důvodů.

Buď se snažíme dosadit hodnotu do funkce.

```
>>> vyraz.subs(x, 0)
0
```

Nebo chceme dosadit část funkce za jinou, která se jí rovná.

```
>>> vyraz = sin(2*x) + cos(2*x)
>>> vyraz.subs(sin(2*x), 2*sin(x)*cos(x))
2*sin(x)*cos(x)) + cos(2*x)
```

Ohledně funkce subs() je důležité poznamenat, že vrací nový výraz, takže použitý výraz nijak nemění.

```
>>> vyraz = sin(x)
>>> vyraz.subs(x, 0)
0
>>> vyraz
sin(x)
>>> x
x
```

(Sympy docs, 2022)

limit():

Sympy využívá funkce limit() na při počítání symbolických limit.

Syntax počtu $\lim_{x \to x_0} f(x)$ je limit(f(x), x, x0).

```
>>> limit((sin(x)/x, x, 0)
1
```

Limita se používá místo subs() v případě, když bod, který chceme zhodnotit, je singularita.

I když Sympy má objekty reprezentující ∞, používat je při zhodnocení není spolehlivé, protože tyto objekty nezohleduplňují například rychlost růstu funkce.

Pro určení limit zprava nebo zleva používáme operátory "+" nebo "-" jako čtvrtý argument funkce limit().

```
>>> limit(1/x, x, 0, '+')
∞
>>> limit(1/x, x, 0, '-')
-∞
```

(Sympy docs, 2022)

Matrix():

Funkce Matrix() vytváří matici.

```
>>> Matrix([[1, 2], [3, 4], [5, 6]])
⎡1  2⎤
⎢3  4⎥
⎣5  6⎦
```

Můžeme vytvářet matice zadáním rozměrů a seznamu.

```
>>> Matrix(2, 3, [1, 2, 3, 4, 5, 6])
⎡1  2  3⎤
⎣4  5  6⎦
```

Dále také existuje několik speciálních konstruktorů pro rychlé vytváření matic: eye() vytvářejí jednotkovou matici, zeros a one vytváří matice plné nul nebo jedniček, a diag posadí matice nebo elementy na diagonálu.

```
>>> eyes(4)
⎡1  0  0  0⎤
⎢0  1  0  0⎥
⎢0  0  1  0⎥
⎣0  0  0  1⎦
>>> zeros(2)
⎡0  0⎤
⎣0  0⎦
>>> ones(2)
⎡1  1⎤
⎣1  1⎦
>>> diag(1, Matrix([[1, 2], [3, 4]]))
⎡1  0  0⎤
⎢0  1  2⎥
⎣0  3  4⎦
```

(Sympy docs, 2022)

S maticemi lze manipulovat stejně jako se seznamy.

m.inv(method="LU"):

Tato funkce vytvoří inverzní matici metodou LU rozkladu.

```
>>> M = Matrix(( [1, 2, 3], [3, 6, 2], [2, 0, 1] ))
>>> M.inv(method="LU")
⎡-13/4   1/14   1/2 ⎤
⎢-1/28   5/28   -1/4⎥
⎣ 3/7    -1/7    0  ⎦
```

(Sympy docs, 2022)

plot():

Funkce plot() vykreslí graf funkce jedné proměnné jako křivku.

Prvním argumentem jsou funkce jedné proměnné, a naposledy vymezíme definiční obor funce třeba takto (x, -5, 5). V případě, že neurčíme definiční obor, funkce ho automatic určí (-10, 10).

Plot() zvládá více funkcí najednou, které se objeví na jednom stejném grafu.

(Sympy docs, 2022)

plot3d():

Funkce plot3d() vykreslí graf v trojrozměrné rovině.

Základní pravidla pro specifikace se takřka neliší od funkce plot(). Jediné, co musíme specifikovat pro daný výraz, jsou definiční obory pro obě proměnné.

Jako u plot() je výchozí interval (-10, 10).

Pro více výrazů s různými definičnímy obory je nutné interval specifikovat.

(Sympy docs, 2022)

2.3 Ostatní knihovny

Při práci s Pythonem lze využívat různé knihovny pro různé situace.

Následující knihovny byly využity v praktické části práce při vykreslování grafů.

2.3.1 Numpy

Numpy je základní knihovna pro vědecké výpočty.

Poskytuje objekt vícerozměrného pole, objekty od tohoto odvozené a sortiment rutin pro rychlé operace s poli.

linspace():

Tato funkce vrací rovnoměrně rozmístěná čísla na specifickém intervalu.

Funkce má tři hlavní argumenty, to jsou *start,* odkud vlastně začínáme, *stop,* kde běžně končíme, a argument *num*, který určuje počet rozmístěných čísel.

Vrací array.

```
>>> import numpy as np
>>> np.linspace(2.0, 3.0, num=5)
array([2.  , 2.25, 2.5 , 2.75, 3.  ])
```

Argumentů má více, jako například endpoint = False, kde argument *stop* není součástí výsledné array.

```
>>> np.linspace(2.0, 3.0, num=5, endpoint = False)
array([2. , 2.2, 2.4, 2.6, 2.8])
```

Nebo třeba retstep = True, kde nám kromě array funkce vrací délku mezery mezi jednotlivými čísly.

```
>>> np.linspace(2.0, 3.0, num=5, retstep = True)
(array([2.  , 2.25, 2.5 , 2.75, 3.  ]), 0.25)
```

Funkce vytváří lineární prostor, který je užitečný při kreslení grafů pomocí knihovny Matplotlib jak uvidíme níže.

(Numpy docs, 2022)

arrange():

Tato funkce dělá takřka to samé, jako funkce linspace.

Vrací tedy rovnoměrně rozmístěná čísla na specifickém intervalu.

Lepší je využívat funkci linspace() v případě kdy jsou mezery mezi čísly datového typu float.

(Numpy docs, 2022)

meshgrid():

Funkce vrací matici tvořenou ze souřadnicových vektorů.

```
>>> nx, ny = (3, 2)
>>> x = np.linspace(0, 1, nx)
>>> y = np.linspace(0, 1, ny)
>>> x
array([0. , 0.5, 1. ])
>>> y
array([0., 1.])
>>> xv, yv = np.meshgrid(x, y)
>>> xv
array([[0. , 0.5, 1. ],
       [0. , 0.5, 1. ]])
>>> yv
array([[0., 0., 0.],
       [1., 1., 1.]])
```

Meshgrid je velmi užitečná funkce, která zhodnotí funkce na souřadnicové síti. Využívá se při konstrukci grafů pomocí knihovny Matplotlib.

(Numpy docs, 2022)

asarray():

Funkce vrací zadaný input ve tvaru array.

Tedy konvertuje data. Tato data mohou být ve tvaru seznamů, seznamů n-tic, n-tic, n-tic m-tic, n-tic seznamů a ndarray.

```
>>>a = [1, 2]
>>> np.asarray(a)
array([1, 2])
```

(Numpy docs, 2022)

2.3.2 Matplotlib

Matplotlib je obsáhlá knihovna, která slouží ke konstrukci statických, animovaných a interaktivních vizualizací v rámci Pythonu.

subplots():

Vytvoří obrazec a soubor podgrafů.

Tento užitečný obal jednoduše povoluje vytvářet běžné rozložení podgrafů, dohromady s ohrazujícím objektem obrazce, jedním povelem.

Vrací objekty třídy Figure a axes.Axes nebo jejich array.

Figure je hlavním kontejnerem pro všechny elementy grafu.

Axes obsahuje většinu elementů, které patří do Figure a vytváří systém používaných souřadnic.

Obvykle se zapisuje v tomto tvaru.

```
>>> import matplotlib.pyplot as plt
>>>fig, ax = plt.subplots()
```

(Matplotlib docs, 2022)

contour():

Funkce vykreslí konturový graf.

Běžně se volá takto.

```
>>> plt.contour([X, Y,] Z, [levels])
```

Kde [X,Y] reprezentují souřadnice hodnot na Z, obě musí být dvojrozměrné a ve stejném tvaru jako Z. Z reprezentuje hodnoty “výšky” ve kterých je graf vykreslen.

A [levels] reprezentuje počet kontur a jejich pozic. Příkazem clabel() se dá konturový graf popsat. Existuje i contourf(), který vytvoří vyplnění konturový graf.

(Matplotlib docs, 2022)

scatter():

Funkce scatter() vrací bodový diagram v různých barvách a velikostech.

Seznam hodnot x, y určí pozice bodů a ostatní argumenty ovlivní velikost a barvu.

(Matplotlib docs, 2022)

plot():

Funkce vykreslí graf souřadnic x a y ve tvaru úseček, nebo značek.

Následujíc příkaz vykreslí graf obyčejně.

```
>>> plt.plot(x, y)
```

Tento příkaz už vykreslí graf značek ve tvaru modrých kruhů

```
>>> plt.plot(x, y, “bo”)
```

Nebo v červených pluskách.

```
>>> plt.plot(x, y, “r+”)
```

(Matplotlib docs, 2022)

show():

Tento příkaz ukáže všechny volně dostpné objekty typu Figure. Tedy všechny grafy dohromady.

3. Úlohy řešené využitím knihovny sympy

3.1 Hlavička

Při práci v Pythonu je nejdůležitější zajistit, aby měl programátor k dispozici všechny potřebné nástoje. Tyto nástroje jsou již předem vytvořené funkce v rámci různých knihoven. Lze je lehce zavolat pomocí funkce import.

Buňku, která obsahuje všechny importované knihovny nebo pouze jejich části, převážně nazýváme hlavička.

In [1]:

```
from math import *
import sympy as sym
from sympy.plotting import plot3d
from IPython.display import Math, display, Latex
import numpy as np
import matplotlib.pyplot as plt
from sympy import *
from sympy.solvers import solve
from sympy.abc import x,y
lam = sym.symbols('lambda')
sym.init_printing()
```

3.2 Lokální extrémy funkcí jedné proměnné

Následující kapitola ukazuje řešení příkladů na **lokální extrémy** funkcí jedné reálné proměnné pomocí Pythonu využitím knihovny sympy.

Definice

Nechť je funkce f definována na okolí bodu a. Řekněme, že tato funkce má v bode a

- **Lokální maximum** $\Leftrightarrow \exists \delta > 0\ \forall x \in (a-\delta, a+\delta): f(x) \leq f(a)$
- **Ostré lokální maximum** $\Leftrightarrow \exists \delta > 0\ \forall x \in (a-\delta, a) \cup (a, a+\delta): f(x) < f(a)$
- **Lokální minimum** $\Leftrightarrow \exists \delta > 0\ \forall x \in (a-\delta, a+\delta): f(x) \geq f(a)$
- **Ostré lokální minimum** $\Leftrightarrow \exists \delta > 0\ \forall x \in (a-\delta, a) \cup (a, a+\delta): f(x) > f(a)$

Lokální minimum a lokální maximum považujeme za ***lokální extrémy***, ostré lokální minimum a ostré lokální maximum za ***ostré lokální extrémy***.

Věta

Nechť má funkce f v bodě a první derivaci. Má-li f v bodě a lokální extrém, pak $f'(x) = 0$.

Tento bod se nazývá *stacionární*.

Lokální extrémy mohou nastat práve v stacionárních bodech, nebo v bodech, kde $f'(x)$ neexistuje.

Věta

Nechť má funkce f v bodě a první a druhou derivaci. Jestliže je první derivace nulová a druhá nenulová, funkce f má v bode a ostrý lokální extrém. Je-li $f''(x) < 0$, jedná se o lokální maximum, je-li $f''(x) > 0$, jedná se o lokální minimum.

Dle předchozích dvou vět lze sestavit postup, jak lokální extrémy funkcí jedné reálné proměnné naleznout.

Nejprve nalezneme všechny stacionární body. V každém znich dále určíme znaménko druhé derivace. Je-li druhá derivace kladná, daná funkce v tomto bode nabývá svého lokálního maxima, v opačném případě se jedná o lokální minimum.

Může se ale stát, že jak první tak druhá derivace je nulová. V tomto případě nemůže o chování funkce poblíž tohoto bodu podle předchozích dvou vět určic nic určitého a musíme použit obecnější větu.

Věta

Platí-li $f'(a) = f''(a) = \cdots = f^{(n)} = 0$ a $f^{(n+1)} \neq 0$, kde *n* je liché, nabývá funkce f v bodě a lokálního extrému: pro $f^{(n+1)} > 0$ lokálního minima a pro $f^{(n+1)} < 0$ lokálního maxima. Je-li naopak n sudé, funkce f v bodě a nemá žádný lokální extrém.

(Kalus, Hřivňák, 2001; Kuben, Došlá, 2004)

3.2.1 Kód v Pythonu

Tato kapitolu ukazuje, jak vypadá funkční kód v pythonu, který spočítá extrém dané funkce jedné reálné proměnné.

In [2]:

```
def najdi_extrem(f, symbol):
    """
    Funkce najdi_extrem vrací extrém zadané funkce f pro vybraný symbol a následně vykreslí graf omezený na okolí podezřelých bodů. Nakonec vykreslí graf funkce f na definičním oboru vybraného symbolu.

    Příklady
    ==========
    Funkce automaticky vyhodnotí o jaký typ extrému se jedná a následně svůj závěr vypíše:

    >>>najdi_extrem(x**2, x)
    Funkce f(x) = x^2 nabývá lokálního minima [0,0]
    Objekt plot obsahující:
    Funkci f(x) v kartézské soustavě souřadnic pro x na intervalu (-1, 1)
    Graf funkce f(x) = x^2
    Objekt plot obsahující:
    Funkci f(x) v kartézské soustavě souřadnic na definičním oboru x

    >>>najdi_extrem(x**2 - 4*x, x)
    Funkce f(x) = x^2 - 4x nabývá lokálního minima [2,-4]
    Objekt plot obsahující:
    Funkci f(x) v kartézské soustavě souřadnic pro x na intervalu (1, 3)
    Graf funkce f(x) = x^2 - 4x
    Objekt plot obsahující:
    Funkci f(x) v kartézské soustavě souřadnic na definičním oboru x

    V Případě více extrémů funkce postupně vrací jednotlivé položky ze seznamu podezřelých bodů a graf omezený na jejich okolí :

    >>>najdi_extrem(x**3 - 2*x**2 + x + 1, x)
    Funkce f(x) = x^3 -2x^2 + x + 1 nabývá lokálního maxima [1/3, 31/27]
    Objekt plot obsahující:
    Funkce f(x) v kártézské soustavě souřadnic pro x na intervalu (-2/3, 4/3)
    Funkce f(x) = x^3 -2x^2 + x + 1 nabývá lokálního minima [1,1]
    Objekt plot obsahující:
    Funkce f(x) v kártézské soustavě souřadnic pro x na intervalu (0, 2)
    Graf funkce f(x) = x^3 -2x^2 + x + 1
    Objekt plot obsahující:
    Funkci f(x) v kartézské soustavě souřadnic na definičním oboru x

    Příklad řešení trigonometrické funkce:
    >>>najdi_extrem(sin(x),x)
    Funkce f(x) = sin(x) nabývá lokálního maxima [π/2 + 2kπ, 1]
    Funkce f(x) = sin(x) nabývá lokálního minima [3/π2 + 2kπ, -1]
    Graf funkce f(x) = sin(x)
    Objekt plot obsahující:
    Funkci f(x) v kartézské soustavě souřadnic na definičním oboru x

    >>>najdi_extrem(sin(x) * cos(x),x)
```

```
    Funkce f(x) = sin(x)cos(x) nabývá lokálního minima [-π/4 + kπ, - 1/2]
    Funkce f(x) = sin(x)cos(x) nabývá lokálního maxima [π/4 + kπ, 1/2]
    Graf funkce f(x) sin(x)cos(x)
    Objekt plot obsahující:
    Funkci f(x) v kartézské soustavě souřadnic na definičním oboru x

    V případě, že funkce extrémy nemá, funkce najdi_extrem to zjistí a následovně vyřeší:

    >>>najdi_extrem(ln((x-2)/(x + 1)),x)
    Funkce f(x) = log(x−2)/(x+1) nemá na svém definičním oboru extrém
    Objekt plot obsahující:
    Funkci f(x) v kartézské soustavě souřadnic pro x na celém definičním oboru

    Parametry
    ==========
    f:
       - funkce jedné reálné proměnné
    symbol:
       - objekt třídy sympy.core.symbol.Symbol
       - jedná se o jediný objekt, pro který řešíme danou funkci
       - pro f(x) je to x

    """
    dy = f.diff(symbol)
    ddy = dy.diff(symbol)
    podezrele_body = solve(dy, symbol)
    dfReal = []
    for bod in podezrele_body:
       if bod.is_real:
            dfReal.append(bod)
    if len(dfReal) != 0:
       for hodnota in dfReal:
          ys = f.subs(x,hodnota)
          if ddy.subs(symbol,hodnota) > 0:
             if isinstance(hodnota, (Mul, sym.core.numbers.Pi)):
                minn = min(dfReal)
                maxx = max(dfReal)
                if maxx - minn == pi:
                   display(Math("\\text{ Funkce } f(x) = %s \; \\text{nabývá lokálního minima } \; [%s
+ 2k\pi,%s]" %(sym.latex(f),sym.latex(hodnota),sym.latex(ys))))
                else:
                   display(Math("\\text{ Funkce } f(x) = %s \; \\text{nabývá lokálního minima } \; [%s
+ k\pi,%s]" %(sym.latex(f),sym.latex(hodnota),sym.latex(ys))))
             else:
                display(Math("\\text{ Funkce } f(x) = %s \; \\text{nabývá lokálního minima } \;
[%s,%s]" %(sym.latex(f),sym.latex(hodnota),sym.latex(ys))))
                sym.plot(f, (x, hodnota - 1, hodnota + 1))
          elif ddy.subs(symbol,hodnota) < 0:
             if isinstance(hodnota, (Mul, sym.core.numbers.Pi)):
                minn = min(dfReal)
                maxx = max(dfReal)
                if maxx - minn == pi:
```

```
                display(Math("\\text{ Funkce } f(x) = %s \; \\text{nabývá lokálního maxima } \; [%s
+ 2k\pi,%s]" %(sym.latex(f),sym.latex(hodnota),sym.latex(ys))))
            else:
                display(Math("\\text{ Funkce } f(x) = %s \; \\text{nabývá lokálního maxima } \; [%s
+ k\pi,%s]" %(sym.latex(f),sym.latex(hodnota),sym.latex(ys))))
        else:
            display(Math("\\text{ Funkce } f(x) = %s \; \\text{nabývá lokálního maxima } \;
[%s,%s]" %(sym.latex(f),sym.latex(hodnota),sym.latex(ys))))
            sym.plot(f, (x, hodnota - 1, hodnota + 1))
    else:
        dddy = ddy.diff(symbol)
        while dddy.subs(symbol,hodnota) == 0:
            dddy = dddy.diff(symbol)
        if dddy.subs(symbol,hodnota) > 0:
            display(Math("\\text{ Funkce } f(x) = %s \; \\text{nabývá lokálního minima } \;
[%s,%s]" %(sym.latex(f),sym.latex(hodnota),sym.latex(ys))))
            sym.plot(f, (x, hodnota - 1, hodnota + 1))
        else:
            display(Math("\\text{ Funkce } f(x) = %s \; \\text{nabývá lokálního maxima } \;
[%s,%s]" %(sym.latex(f),sym.latex(hodnota),sym.latex(ys))))
            sym.plot(f, (x, hodnota - 1, hodnota + 1))
    display(Math("\\text{Graf funkce } f(x) = %s" %(sym.latex(f))))
    sym.plot(f)
else:
    display(Math("\\text{ Funkce } f(x) = %s \\text{
nemá na svém definičním oboru extrém}" %(sym.latex(f))))
    sym.plot(f)
```

(Vlastní zpracování v softwaru Jupyter Notebook; Kalus, Hřivňák, 2001; The Organic Chemistry Tutor, 2018)

3.2.2 Testování

Kód byl důkladně otestován na následujících příkladech.

Určete lokální extrémy funkce:

a) $y = x^3 - 2x^2 - 4x + 5,$

b) $y = \frac{x^2}{x+4}$

c) $y = e^{x^2+2x-4},$

d) $y = ln\frac{x-2}{x+1}$

e) $y = \arctan x^2 + 1,$

f) $y = \sin x \cos x$

g) $y = \arcsin\frac{x}{x+1},$

h) $y = \frac{3}{4} ln\frac{x+2}{2-x} - x$

Výsledky:

a) max $[-\frac{2}{3},\frac{175}{27}]$, min $[2,-3]$; b) max $[-8,-16]$, min $[0,0]$; c) min $[-1,e^{-5}]$; d) nemá extrémy; e) min $[0,\frac{\pi}{4}]$; f) max $[\frac{\pi}{4}+k\pi,12]$, min $[-\frac{\pi}{4}+k\pi,-12]$; g) nemá extrémy; h) max $[-1,0.18]$, min $[1,-0.18]$. (Hamříková, 2007)

3.2.3 Příklad

Nalezněte lokální extrémy funkce $f(x)=x^3-2x^2+x+1$ na intervalu $(-\infty,\infty)$.

Řešení:

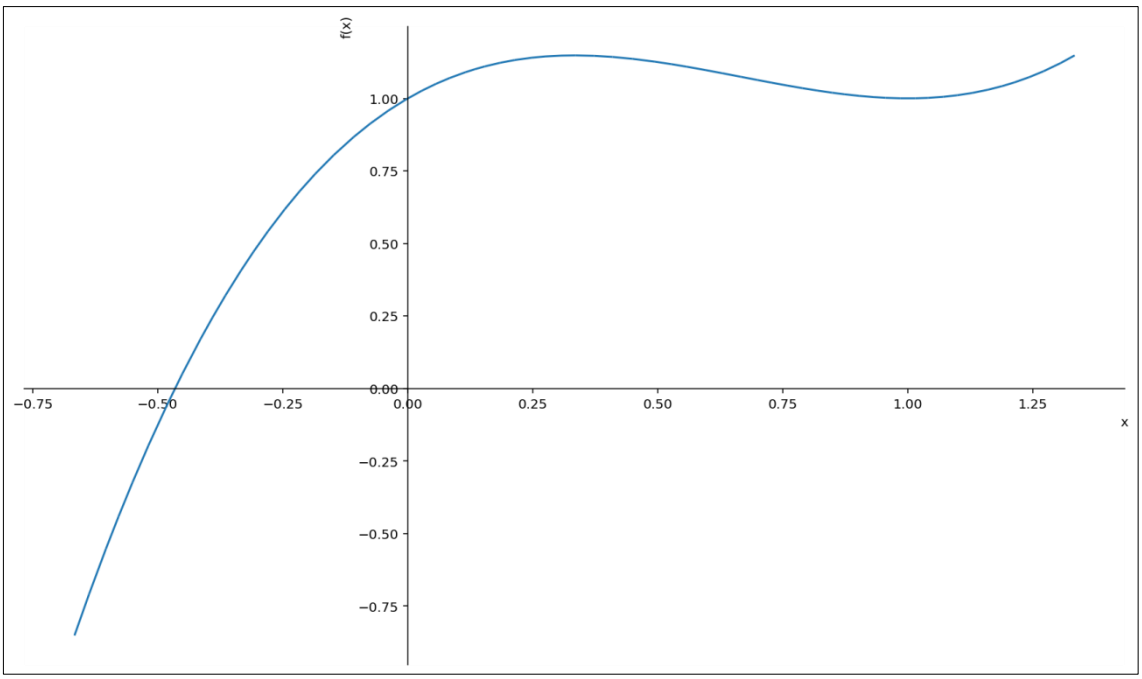
In [3]:

```
najdi_extrem(x**3 - 2*x**2 + x + 1, x)
```

Out [3]:

Funkce $f(x)=x^3-2x^2+x+1$ nabývá lokálního maxima pro $x=\frac{1}{3}$

Obrázek č. 1 – Graf funkce $f(x)=x^3-2x^2+x+1$ v okolí bodu $x=\frac{1}{3}$

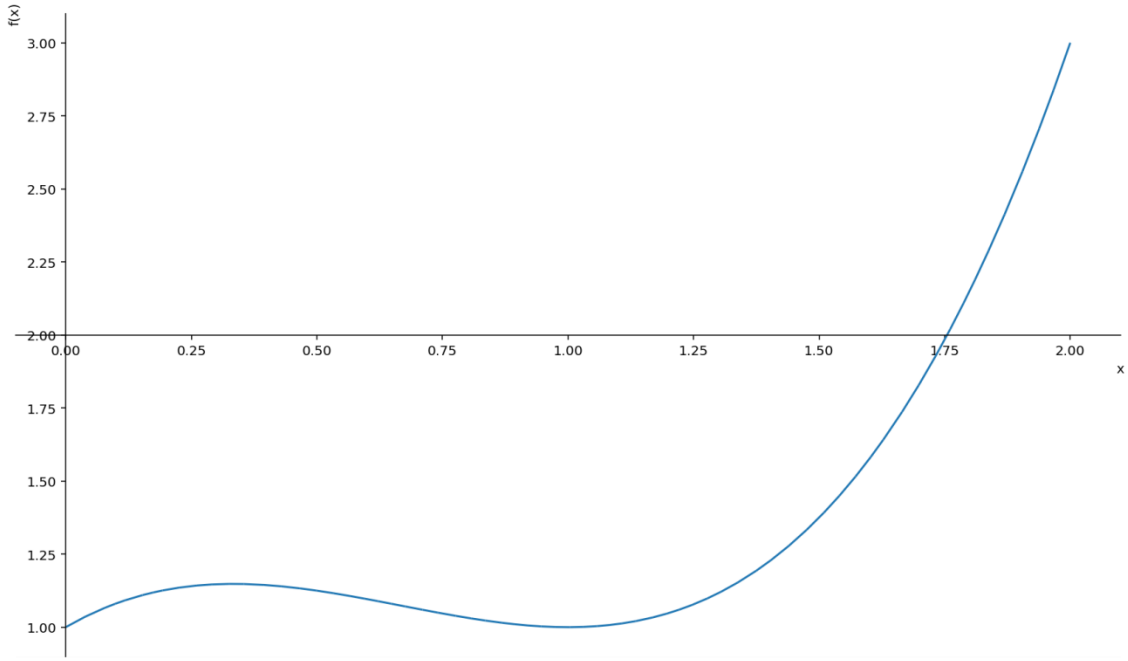


Zdroj: Vlastní zpracování v programu Jupyter Notebook

Out [3]:

Funkce $f(x) = x^3 - 2x^2 + x + 1$ nabývá lokálního minima pro $x = 1$

Obrázek č. 2 – Graf funkce $f(x) = x^3 - 2x^2 + x + 1$ v okolí bodu $x = 1$

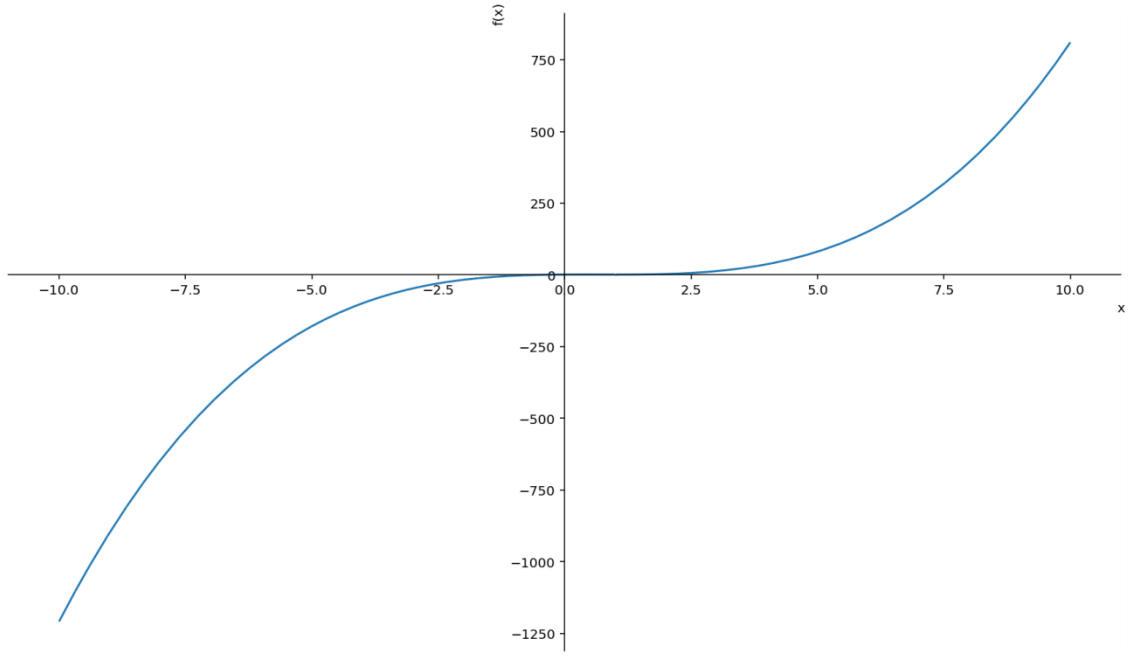


Zdroj: Vlastní zpracování v programu Jupyter Notebook

Out [3]:

Graf funkce $f(x) = x^3 - 2x^2 + x + 1$

Obrázek č. 3 – Graf funkce $f(x) = x^3 - 2x^2 + x + 1$



Zdroj: Vlastní zpracování v programu Jupyter Notebook

3.3 Globální extrémy funkcí jedné proměnné

Následující kapitola ukazuje řešení příkladů na **globální extrémy** funkcí jedné reálné proměnné pomocí Pythonu využitím knihovny sympy.

Definice

Nechť je funkce f definována na podmnožině reálných čísel $A \subset \mathbb{R}$. Řekněme, že funkce f nabývá v bodě $a \in A$

- **Maxima na A** $\Leftrightarrow \forall x \in A\colon\ f(x) \leq f(a)$
- **Ostrého maxima na A** $\Leftrightarrow \forall x \in A\backslash\{a\}\colon f(x) < f(a)$
- **Minima na A** $\Leftrightarrow \forall x \in A\colon\ f(x) \geq f(a)$
- **Ostrého minima na A** $\Leftrightarrow \forall x \in A\backslash\{a\}\colon\ f(x) < f(a)$

Maxima a minima funkce f na množině A obvykle označujeme názvem **globální extrémy.**

Extrémy funkce zpravidla vyšetřujeme na nějakém intervalu. Exitence ani jednoho z extrému však není na daném intervalu zaručená. Pro uzavřené intervaly funkce naopak nabývá globálních extrémů vždy.

Při hledání extrémů funkce na uzavřeném intervalu $\langle a, b\rangle$ zřejmě musíme hledat buď v krajních bodech a, b nebo uvnitř daného intervalu, tedy v bodech otevřeného intervalu (a, b).

Pro zjištění globálních extrémů musíme nejprve znát všechny body podezřelé z extrémů. To znamená

- Krajní body intervalu
- Body, v nichž má studovaná funkce nulovou derivaci
- Body, v nichž funkce derivaci nemá nebo je nespojitá

Těchto bodu bývá zpravidla konečně mnoho, převážně jich je pouze několik.

Po zjištění všech podezřelích bodů stačí zjistit funkční hodnotu funkce v těchto bodech. Následující rozhodnutí je již snadné, protože v bodě, kde má funkce nejvyšší funkční hodnotu, nabývá funkce **globálního maxima** a v bodě, kde má nejnižší funkční hodnotu, nabývá **globálního minima**.

Hledání globálních extrému funkce na polootevřeném nebo otevřeném intervalu postup se v některých částech mění.

Nejprve z množiny podezřelích bodů musíme vyloučit krajní body intervalu, které do něj nepatří. Ostatní body zachováme.

Jako v předchozím postupu nalezneme v těchto bodech funkční hodnoty a porovnáme je.

Tyto body neporovnáme jenom mezi sebou, ale musíme je take porovnat s odpovídajícími jednostrannými limitami funkce v krajních bodech, které do intervalu nepatří. V případě, že tyto limity existují.

Jesltiže bude největší z těchto limit větší než největší z funkčních hodnot vyšetřovaných bodů, funkce svého maxima na studovaném intervalu nenabývá. Naopak bude-li největší z těchto limit menší než největší funkční hodnota, funkce svého maxima na studovaném intervalu nabývat bude. Podobné závěry lze formulovat i pro existence minima.

(Kalus, Hřivňák, 2001; Kuben, Došlá, 2004)

3.3.1 Kód v Pythonu

Následující kód v pythonu spočitá globální extrémy dané funkce využitím převážně knihovny sympy.

In [4]:

```
def najdi_globalni_extrem(f,symbol,dolni_mez,horni_mez):
    """
    Funkce najdi_globalni_extrem vrací extrémy zadané funkce f na omezeném intervalu a následné vykreslí graf

    Příklady
    ==========
    Funkce se pomocí parametrů dolni_mez a horni_met orientuje a na omezeném intervalu najde extrémy. Poté zjistí, zda se jedná o globální minimum nebo maximum. Nejprve však
```

musíme zodpovědět na otázku, jak je omezující interval uzavřen:
Rozhodněte, zda je funkce uzavřená. Odpovězte Ano, Ne, Zleva nebo Zprava:

Pří zodpovězení Ano funkce najdi_globalni_extrem hledá globální extrémy na uzavřeném intervalu

>>>najdi_globalni_extrem(x**2 - 3*x + 4, x, 0, 2)
Rozhodněte, zda je funkce řešena na uzavřeném intervalu. Odpovězte Ano, Ne, Zleva nebo Zprava: Ano
Funkce f(x) = x^2 - 3x + 4 omezená na intervalu <0,2> nabývá globální maximum [0,4] a globální minimum [3/2,7/4]
Objekt plot obsahující:
Funkci f(x) v kartézské soustavě souřadnic pro x na intervalu <0, 2>

>>>najdi_globalni_extrem(cos(x) + sin(x),x,0,2*pi)
Rozhodněte, zda je funkce řešena na uzavřeném intervalu. Odpovězte Ano, Ne, Zleva nebo Zprava: Ano
Funkce f(x) = cos(x) + sin(x) omezená na intervalu <0,2π> nabývá globální maximum [π/4,√2] a globální minimum [5π/4,-√2]
Objekt plot obsahující:
Funkci f(x) v kartézské soustavě souřadnic pro x na intervalu <0, 2π>

Jestliže zvolíme Ne, pak funkce najdi_globalni_extrem považuje omezující interval za otevřený

>>>najdi_globalni_extrem(tan(x) - 2*x,x,-pi/2,pi/2)
Rozhodněte, zda je funkce řešena na uzavřeném intervalu. Odpovězte Ano, Ne, Zleva nebo Zprava: Ne
Funkce f(x) = -2x + tan(x) omezená na intervalu (-π/2, π/2) nemá globální extrém
Objekt plot obsahující:
Funkci f(x) v kartézské soustavě souřadnic pro x na intervalu (-π/2, π/2)

>>>najdi_globalni_extrem(-x**3 + 3*x + 1, x, -3,5)
Rozhodněte, zda je funkce řešena na uzavřeném intervalu. Odpovězte Ano, Ne, Zleva nebo Zprava: Ne
Funkce f(x) = -x^3 + 3x + 1 omezená na intervalu (-3,5) nabývá globální maximum [1,3] a globální minimum [-1,-1]
Objekt plot obsahující:
Funkci f(x) v kartézské soustavě souřadnic pro x na intervalu (-3, 5)

Pro funkci uzavřenou Zleva:

>>>najdi_omezeny_extrem(x + 4/x + 5, x, -5, 0)
Rozhodněte, zda je funkce řešena na uzavřeném intervalu. Odpovězte Ano, Ne, Zleva nebo Zprava: Zleva
Funkce f(x) = x + 4/x + 5 omezená na intervalu <-5,0) nabývá globální maximum [-2,1] a nenabývá globální minimum
Objekt plot obsahující:
Funkci f(x) v kartézské soustavě souřadnic pro x na intervalu <-5, 0)

>>>najdi_globalni_extrem(x - (1/(x+1)),x,-4,-1)
Rozhodněte, zda je funkce řešena na uzavřeném intervalu. Odpovězte Ano, Ne, Zleva nebo

```
Zprava: Zleva
   Funkce f(x) = x - 1/x+1 omezená na intervalu <-4,-1) nemá globální maximum a má globální
minimum [-4,- 11/3]
   Objekt plot obsahující:
   Funkci f(x) v kartézské soustavě souřadnic pro x na intervalu <-4, -1)

   Zprava:

   >>>najdi_globalni_extrem(sqrt(x) + (4/x) -2,x, 0,7)
   Rozhodněte, zda je funkce řešena na uzavřeném intervalu. Odpovězte Ano, Ne, Zleva nebo
Zprava: Zprava
   Funkce f(x) = sqrt(x) + 4/x -2 omezená na intervalu (0, 7> nenabývá globální maximum a
nabývá globální minimum [4,1]
   Objekt plot obsahující:
   Funkci f(x) v kartézské soustavě souřadnic pro x na intervalu (0, 7>

   >>>najdi_globalni_extrem(atan(x), x, -1, 1)
   Rozhodněte, zda je funkce řešena na uzavřeném intervalu. Odpovězte Ano, Ne, Zleva nebo
Zprava: Zprava
   Funkce f(x) = atan(x) omezená na intervalu (-1,1> nabývá globální maximum [1,π/4] a
nenabývá globální minimum


   Parametry
   ==========

   f:
      - funkce jedné reálné proměnné
   symbol:
      - objekt třídy sympy.core.symbol. Symbol
      - jedná se o jediný objekt, pro který řešíme danou funkci
      - pro f(x) je to x
   dolni_mez:
      - může obsahovat int, float nebo i algebraický výraz
      - je to spodní hranice intervalu omezující funkci f
   horni_mez:
      - může obsahovat int, float nebo i algebraický výraz
      - je to horní hranice intervalu omezující funkci f
   """
   uzavrenost = input("Rozhodněte, zda je funkce řešena na uzavřeném intevalu. Odpovězte
Ano, Ne, Zleva nebo Zprava: ")
   dy = f.diff(symbol)
   ddy = dy.diff(symbol)
   podezrele_body = solveset(dy, symbol,domain=Interval(dolni_mez,horni_mez))
   if uzavrenost == "Ano" :
      hodnoty = [dolni_mez,horni_mez]
      hodnoty.extend(podezrele_body)
      y_list = []
      for bod in hodnoty:
         y = f.subs(symbol, bod)
         y_list.append(y)
      glob_min = min(y_list)
```

```
        glob_max = max(y_list)
        x_min = hodnoty[y_list.index(glob_min)]
        x_max = hodnoty[y_list.index(glob_max)]
        display(Math("\\text{Funkce} \; f(x) = %s \; \\text{omezená na intervalu} \; \langle%s, %s
\\rangle \; \\text{nabývá globální maximum } \; [%s,%s] \; \\text{a globální minimum } \;
[%s,%s] "
%(sym.latex(f),sym.latex(dolni_mez),sym.latex(horni_mez),sym.latex(x_max),sym.latex(glob_
max),sym.latex(x_min), sym.latex(glob_min))))
    elif uzavrenost == "Ne":
        hodnoty = []
        hodnoty.extend(podezrele_body)
        y_list = []
        for bod in hodnoty:
            y = f.subs(symbol, bod)
            y_list.append(y)
        glob_min = min(y_list)
        glob_max = max(y_list)
        x_min = hodnoty[y_list.index(glob_min)]
        x_max = hodnoty[y_list.index(glob_max)]
        if limit(f,symbol,horni_mez, "-") == oo and limit(f,symbol,dolni_mez, "+") == oo:
            display(Math("\\text{Funkce} \; f(x) = %s \; \\text{omezená na intervalu} \; (%s, %s) \;
\\text{nenabývá globální maximum a nabývá globální minimum } \; [%s,%s] "
%(sym.latex(f),sym.latex(dolni_mez),sym.latex(horni_mez),sym.latex(x_min),sym.latex(glob_m
in))))
        elif limit(f,symbol,horni_mez, "-") == -oo and limit(f,symbol,dolni_mez, "+") == -oo:
            display(Math("\\text{Funkce} \; f(x) = %s \; \\text{omezená na intervalu} (%s, %s) \;
\\text{nabývá globální maximum } \; [%s,%s] \; \\text{a nenabývá globální minimum} "
%(sym.latex(f),sym.latex(dolni_mez),sym.latex(horni_mez), sym.latex(x_max),
sym.latex(glob_max))))
        elif limit (f,symbol,horni_mez, "-") == oo and limit(f,symbol,dolni_mez, "+") == -oo:
            display(Math("\\text{Funkce} \; f(x) = %s \; \\text{omezená na intervalu} \; (%s, %s) \;
\\text{nemá globální extrém}" %((sym.latex(f),sym.latex(dolni_mez),sym.latex(horni_mez)))))
        elif limit (f,symbol,horni_mez,"-") == -oo and limit(f,symbol,dolni_mez,"+") == oo:
            display(Math("\\text{Funkce} \; f(x) = %s \; \\text{omezená na intervalu} \; (%s, %s) \;
\\text{nemá globální extrém}" %((sym.latex(f),sym.latex(dolni_mez),sym.latex(horni_mez)))))
        else:
            display(Math("\\text{Funkce} \; f(x) = %s \; \\text{omezená na intervalu} (%s, %s) \;
\\text{nabývá globální maximum} [%s,%s] \; \\text{a globální minimum} \; [%s,%s] "
%(sym.latex(f),sym.latex(dolni_mez),sym.latex(horni_mez),sym.latex(x_max),sym.latex(glob_
max),sym.latex(x_min),sym.latex(glob_min))))
    elif uzavrenost == "Zleva":
        hodnoty = [dolni_mez]
        hodnoty.extend(podezrele_body)
        y_list = []
        for bod in hodnoty:
            y = f.subs(symbol, bod)
            y_list.append(y)
        glob_min = min(y_list)
        glob_max = max(y_list)
        x_min = hodnoty[y_list.index(glob_min)]
        x_max = hodnoty[y_list.index(glob_max)]
        if limit(f,symbol,horni_mez, "-") == oo:
```

```
            display(Math("\\text{Funkce} \; f(x) = %s \; \\text{omezená na intervalu} \langle%s, %s) \; \\text{nenabývá globální maximum a nabývá globální minimum} \; [%s,%s] "
%(sym.latex(f),sym.latex(dolni_mez),sym.latex(horni_mez),sym.latex(x_min),sym.latex(glob_min))))
        elif limit(f,symbol,horni_mez, "-") == -oo:
            display(Math("\\text{Funkce} \; f(x) = %s \; \\text{omezená na intervalu} \langle%s, %s) \; \\text{nabývá globální maximum} \; [%s,%s] \; \\text{a nenabývá globální minimum} "
%(sym.latex(f),sym.latex(dolni_mez),sym.latex(horni_mez),
sym.latex(x_max),sym.latex(glob_max))))
        else:
            if len(hodnoty) == 1:
                fs1 = f.subs(symbol, dolni_mez)
                fs2 = f.subs(symbol, dolni_mez + 1)
                if fs1 < fs2:
                    display(Math("\\text{Funkce} \; f(x) = %s \; \\text{omezená na intervalu} \; \langle%s, %s) \; \\text{nenabývá globální maximum a nabývá globální minimum} \; [%s,%s] "
%(sym.latex(f),sym.latex(dolni_mez),sym.latex(horni_mez),sym.latex(x_min),sym.latex(glob_min))))
                else:
                    display(Math("\\text{Funkce} \; f(x) = %s \; \\text{omezená na intervalu} \; \langle%s, %s) \; \\text{nabývá globální maximum} \; [%s,%s] \; \\text{a nenabývá globální minimum} " %(sym.latex(f),sym.latex(dolni_mez),sym.latex(horni_mez),
sym.latex(x_max),sym.latex(glob_max))))
            else:
                display(Math("\\text{Funkce} \; f(x) = %s \; \\text{omezená na intervalu} \; \langle%s, %s) \; \\text{nabývá globální maximum} \; [%s,%s] \; \\text{a globální minimum} \; [%s,%s] "
%(sym.latex(f),sym.latex(dolni_mez),sym.latex(horni_mez),sym.latex(x_max),sym.latex(glob_max),sym.latex(x_min),sym.latex(glob_min))))
    elif uzavrenost == "Zprava":
        hodnoty = [horni_mez]
        hodnoty.extend(podezrele_body)
        y_list = []
        for bod in hodnoty:
            y = f.subs(x, bod)
            y_list.append(y)
        glob_min = min(y_list)
        glob_max = max(y_list)
        x_min = hodnoty[y_list.index(glob_min)]
        x_max = hodnoty[y_list.index(glob_max)]
        if limit(f,symbol,dolni_mez, "+") == oo:
            display(Math("\\text{Funkce} \; f(x) = %s \; \\text{omezená na intervalu} \; (%s, %s\\rangle \; \\text{nenabývá globální maximum a nabývá globální minimum} \; [%s,%s] "
%(sym.latex(f),sym.latex(dolni_mez),sym.latex(horni_mez),sym.latex(x_min),sym.latex(glob_min))))
        elif limit(f,symbol,dolni_mez, "+") == -oo:
            display(Math("\\text{Funkce} \; f(x) = %s \; \\text{omezená na intervalu} \; (%s, %s\\rangle \; \\text{nabývá globální maximum} \; [%s,%s] \; \\text{a nenabývá globální minimum} " %(sym.latex(f),sym.latex(dolni_mez),sym.latex(horni_mez),
sym.latex(x_max),sym.latex(glob_max))))
        else:
            if len(hodnoty) == 1:
                fs1 = f.subs(symbol, horni_mez)
```

```
            fs2 = f.subs(symbol, horni_mez - 1)
            if fs1 < fs2:
                display(Math("\\text{Funkce} \; f(x) = %s \; \\text{omezená na intervalu} \; (%s,
%s\\rangle \; \\text{nenabývá globální maximum a nabývá globální minimum} \; [%s,%s]"
%(sym.latex(f),sym.latex(dolni_mez),sym.latex(horni_mez),sym.latex(x_min),sym.latex(glob_m
in))))
            else:
                display(Math("\\text{Funkce} \; f(x) = %s \; \\text{omezená na intervalu} \; (%s,
%s\\rangle \; \\text{nabývá globální maximum} \; [%s,%s] \\text{ a nenabývá globální
minimum}"
%(sym.latex(f),sym.latex(dolni_mez),sym.latex(horni_mez),sym.latex(x_min),sym.latex(glob_m
in))))
        else:
            display(Math("\\text{Funkce} \; f(x) = %s \; \\text{omezená na intervalu} \; (%s,
%s\\rangle \; \\text{nabývá globální maximum} [%s,%s]  \; \\text{a globální minimum} \;
[%s,%s]  "
%(sym.latex(f),sym.latex(dolni_mez),sym.latex(horni_mez),sym.latex(x_max),sym.latex(glob_
max),sym.latex(x_min),sym.latex(glob_min))))
    else:
        display(Math("\\text{Error}"))
    sym.plot(f, (x, dolni_mez, horni_mez))
```

(Vlastní zpracování v softwaru Jupyter Notebook; Kalus, Hřivňák, 2001; The Organic Chemistry Tutor, 2018)

3.3.2 Testování

Kód byl důkladně otestován na následujících příkladech.

Určete globální extrémy funkce:

a) $y = x^2 - 4x + 3, I = \langle -6,9 \rangle$

b) $y = x - \frac{1}{x-1}, I = \langle -4,-1 \rangle$

c) $y = x^3 + 3x^2 - 9x - 3, I = \langle -4,4 \rangle$

d) $y = \sqrt{x^2 - 4x + 5}, I = \langle -3,0 \rangle$

g) $y = x + \sin 2x, I = \langle \frac{\pi}{4}, \pi \rangle$

f) $y = e^{x^2+2x-3}, I = \langle -2,1 \rangle$

g) $y = x^2 e^{-x}, I = \langle -\frac{1}{2}, 3 \rangle$

h) $y = \arctan x^2 - x - 2, I = \langle -1,1 \rangle$

Výsledky:

a) max $[-6{,}63]$, min $[2,-1]$; b) max nemá, min $[-4,-\frac{11}{3}]$; c) max $[4{,}73]$, min $[1,-8]$; d) max $[-3,\sqrt{26}]$, min $[0,\sqrt{5}]$; e) max $[\pi,3.14]$, min $[\frac{2\pi}{3},1.22]$; f) max $[1,1]$, min $[-1,e^{-4}]$; g) max $[2,4e^{-2}]$, min $[0,0]$; h) max $[-1,0]$, min $[\frac{1}{2},-1.15]$. (Hamříková, 2007)

3.3.3 Příklad

Určete globální extrémy funkce $f(x) = x + \sin 2x$ na intervalu $\langle \frac{\pi}{4}, \pi \rangle$

Řešení:

In [5]:

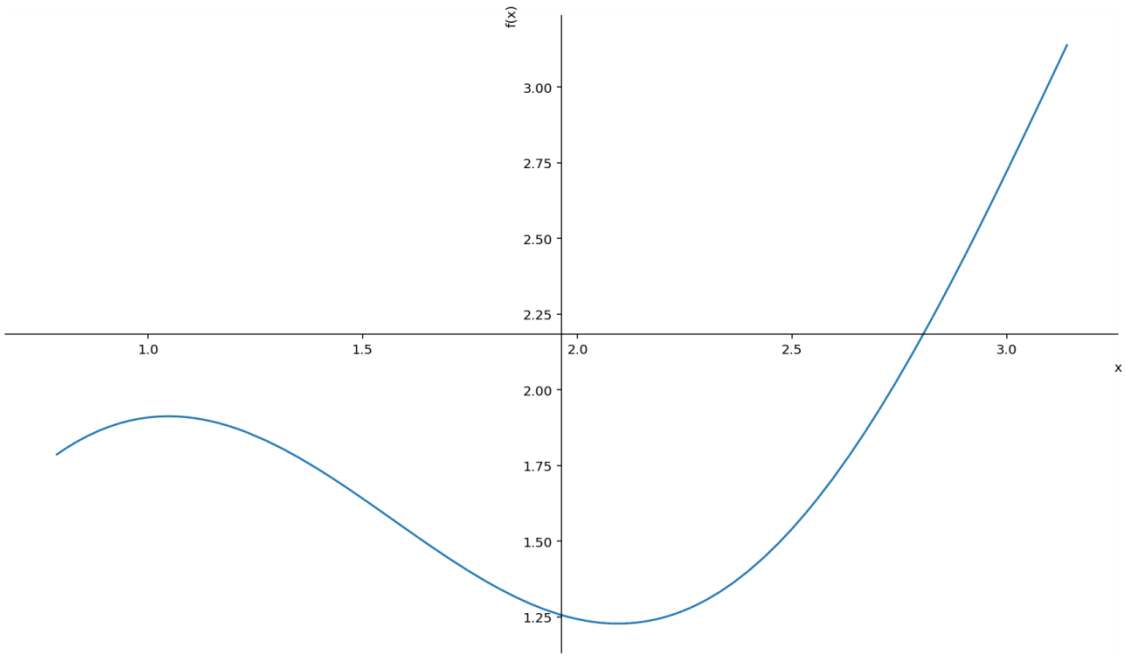
```
najdi_globalni_extrem(x + sin(2*x), x, pi/4, pi)
```

Rozhodněte, zda je funkce řešena na uzavřeném intevalu. Odpovězte Ano, Ne, Zleva nebo Zprava: Ano

Out [5]:

Funkce $f(x) = x + \sin(2x)$ omezená na interval $\langle \frac{\pi}{4}, \pi \rangle$ nabývá globální maximum $[\pi, \pi]$a globální minimum $\left[\frac{2\pi}{3}, -\frac{\sqrt{3}}{2} + \frac{2\pi}{3}\right]$

Obrázek č. 4 – Graf funkce $f(x) = x + \sin(2x)$ omezené na intervalu $\langle \frac{\pi}{4}, \pi \rangle$



Zdroj: Vlastní zpracování v programu Jupyter Notebook

3.4 Lokální extrémy funkcí více proměnných

Definice

Nechť je funkce $f(x)$ definována na nějakém okolí bodu a. Řekněme, že tato funkce má v bodě a

- **Lokální maximum** $\Leftrightarrow \exists \delta > 0 \; \forall x \in D_f : \big||x-a|\big| < \delta \Rightarrow f(x) \leq f(a)$,
- **Ostré lokální maximum** $\Leftrightarrow \exists \delta > 0 \; \forall x \in D_f : \big||x-a|\big| < \delta \Rightarrow f(x) < f(a)$,
- **Lokální minimum** $\Leftrightarrow \exists \delta > 0 \; \forall x \in D_f : \big||x-a|\big| < \delta \Rightarrow f(x) \geq f(a)$,
- **Ostré lokální minimum** $\Leftrightarrow \exists \delta > 0 \; \forall x \in D_f : \big||x-a|\big| < \delta \Rightarrow f(x) > f(a)$,

Lokální maximum a lokální minimum nazýváme **lokální extrémy.** Ostré lokální maximum a ostré lokální minimum nazýváme **ostré lokální extrémy.**

Věta

Nechť má funkce f v okolí bodu a všechny první parciální derivace. Jestliže funkce v bodě a nabývá lokálního extrému, pak se všechny tyto derivace rovnají nule.

Pro spočítání lokálních extrémů funkce vice proměnných nejprve musíme znát body podezřelé z extrémů.

Jak lze z předchozí věty vyčíst, pro získání těchto bodů je nutné provést první parciální derivace funkce f. Tyto parciální derivace položíme nule a získáme tak soustavu rovnic.

Vyřešením soutavy teprve získáme body podezřélé z extrémů.

Na určení, zda se opravdu jedná o extrémy a o jaké extrémy se vlastně jedná, je potřeba **Hessova matice**.

Hessova matice představuje čtvercovou matici druhých parciálních derivací skalární funkce.

Za předpokladu, že existují všechny parciální derivace druhého řádu funkce $f(x_1, x_2, \cdots, x_n)$ má Hessova matice následující tvar:

$$H(f) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

Po získání Hessovy matice dané funkce dosadíme výsledek soustavy rovnic.

Následně podle Sylvestrova kritéria zjistíme, o jaký druh matice se jedná.

Definice Sylvestrovo kritérium

Matice A je

- Pozitivně definitní, jestliže všechny hlavní subdeterminanty matice A jsou kladné:

$$a_{11} > 0, \det\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} > 0, \det\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} > 0, \cdots, \det A > 0;$$

- Negativně definitní, jestliže hlavní subdeterminanty střídají znamínka počínaje záporným:

$$a_{11} < 0, \det\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} > 0, \det\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} < 0, \cdots, (-1)^n \det A > 0;$$

- Indefinitní, jestliže $\det A \neq 0$ a přitom neplatí první dvě pravidla

V případě, že je naše matice po dosazení podezřelého bodu **pozitivně definitní**, jedná se o **ostré lokální maximum**. Jestliže je **negativně definitní** jedná se o ostré **lokální minimum**. Je-li **indefinitní**, pak je podezřelí bod **sedlovým bodem**.

(Kalus, Hřivňák, 2001; Hamhalter, Tišer, 2005)

3.4.1 Kód v Pythonu

Kód byl rozdělene do několika různých funkcí, z nichž hlavní funkce **najdi_ex_vice_prom** využívá ostatní na výpočet lokální extrémů funkcí dvou reálných proměnných.

Kód Hessovy matice

Následující kód v pythonu nám vrací tvar Hessovy matice funkce f.

In [6]:

```
def Hess_Mat(f, symbol1, symbol2):
    """
    Vrací Hessovu matici.

    Příklady
    ==========

    >>>Hess_Mat(x**2 - y**2,x,y)
    [2   0]
    [0  -2]

    >>>Hess_Mat(x**4 + y**4 -(x + y)**2,x,y)
    [12x^2 -2  -2]
    [-2  12y^2 -2]

    >>>Hess_Mat(x*y + 50/x + 20/y,x,y)
    [100/x^3   1]
    [1    40/y^2]

    >>>Hess_Mat(3*x + 2*y - x**3 - y**2,x,y)
    [-6x   0]
    [0    -2]

    Parametry
    ==========

    f:
       - Funkce o dvou reálných proměnných; převážně f(x,y)
    symbol1 :
       - objekt třídy sympy.core.symbol. Symbol
       - jedná se o objekt, pro který řešíme danou funkci
       - pro f(x,y) je to x
    symbol2:
       - objekt třídy sympy.core.symbol. Symbol
       - jedná se o objekt, pro který řešíme danou funkci
       - pro f(x,y) je to y

    """
    gradf = derive_by_array(f, [symbol1,symbol2])
    Hmat = derive_by_array(gradf, [symbol1,symbol2])
    return Hmat
```

In [7]:

```
Hess_Mat(x**4 + y**4 -(x + y)**2,x,y)
```

Out [7]:

$$\begin{bmatrix} 12x^2 - 2 & -2 \\ -2 & 12y^2 - 2 \end{bmatrix}$$

Kód rozhodujících podmínek

Následující dvě funkce rozhodují, zda se jedná o pozitivně definitní či negativně definitní matici.

In [8]:

```
def is_positive(M):
    """
    Tatu funkce vyhodnotí dle podmínek zda se jedná o ostré lokální minimum.

    Příklady
    ==========

    >>>is_positive(Array([[4/5,1],[1,5]]))
    True

    >>>is_positive(Array([[-6, 0],[0,-2]]))
    False

    Parametry
    ==========
    M:
       - objekt třídy ImmutableDenseNDimArray, zkráceně Array
       - Matice
       - Představuje Hessovu matici po dosazení stacionárních bodů.
    """
    if M[0,0] > 0 and M[0,0]*M[1,1] - M[0,1]*M[1,0] > 0:
        return True
    else:
        return False
#########################################################
def is_negative(M):
    """
    Tato funkce vyhodnotí dle podmínek, zda se jedná o ostré lokální maximum.
    Příklady
    ==========

    >>>is_negative(Array([[-54,-54],[-54,-84]]))
```

```
    True

    >>>is_negative(Array([[10,-2],[-2,10]]))
    False

    Parametry
    ==========
    M:
       - objekt třídy ImmutableDenseNDimArray, zkráceně Array
       - Matice
       - Představuje Hessovu matici po dosazení stacionárních bodů.
    """
    if M[0,0] < 0 and M[0,0]*M[1,1] - M[0,1]*M[1,0] > 0:
       return True
    else:
       return False
```

Kód na výpočet extrémů

Tento kód již najde extrémy.

In [9]:

```
def najdi_lok_ext_vice_prom(f,symbol1,symbol2):
    """
    Funkce vrací lokální extrémy funkce více proměnných

    Příklady
    ==========

    >>>najdi_lok_ext_vice_prom(x**2 + y**2,x,y)
    Hessova matice v obecném bodě je rovna matici:

    [2   0]
    [0   2]

    Tady je Hessova matice vyčíslená v bodě (0,0):

    [2   0]
    [0   2]

    Funkce nabývá v bodě (0, 0) ostré lokální minimum 0
    Objekt plot3d obsahující
    Funkci f(x,y) = x^2 + y^2 v kártézské soustavě souřadnic pro x(-1,1); y(-1,1); f(0,2)

    >>>najdi_lok_ext_vice_prom(x**3 + y**3 + 9*x*y + 27,x,y)
    Hessova matice v obecném bodě je rovna matici:

    [6x   9]
```

```
   [9   6y]

   Tady je Hessova matice vyčíslená v bodě(−3, −3):

   [-18   9]
   [9   -18]

   Funkce nabývá v bodě (−3, −3) ostré lokální maximum 54

   Tady je Hessova matice vyčíslená v bodě (0, 0):

   [0   9]
   [9   0]

   Bod (0, 0) je sedlovým bodem funkce
   Objekt plot3d obsahující
   Funkci f(x,y) = x^3 + y^3 + 9xy + 27 v kártézské soustavě souřadnic pro x(-3,0); y(-3,0); f(0,54)

   Parametry
   ==========
   f:
      - Funkce o dvou reálných proměnných; převážně f(x,y)
   symbol1 :
      - objekt třídy sympy.core.symbol.Symbol
      - jedná se o objekt, pro který řešíme danou funkci
      - pro f(x,y) je to x
   symbol2:
      - objekt třídy sympy.core.symbol.Symbol
      - jedná se o objekt, pro který řešíme danou funkci
      - pro f(x,y) je to y
   """
   gradf = derive_by_array(f, [symbol1,symbol2])
   df = nonlinsolve(gradf, [symbol1,symbol2])
   dfReal = []
   neost_ex = []
   for koren in df:
      if koren[0].is_real and koren[1].is_real:
         dfReal.append(koren)
   while len(dfReal) == 0:
      df = solve(gradf, [symbol1,symbol2], rational = True)
      for koren in df:
         if koren[0].is_real and koren[1].is_real:
            dfReal.append(koren)
         elif isinstance(koren[1], Symbol):
            neost_ex.append(koren[0])
   if len(neost_ex) != 0:
      display(Math("\\text{Funkce nabývá na x =  } \; %s \; \\text{lokálního neostrého extrému.
Charakter extrému lze jednoduše určit pomocí grafu Contour.} " %(sym.latex(neost_ex))))
   display(Math("\\text{Hessova matice v obecném bodě je rovna matici: }"))
   display(Hess_Mat(f, symbol1, symbol2))
   for stac_bod in dfReal:
      dosazH=Hess_Mat(f,symbol1, symbol2).subs([(symbol1, stac_bod[0]),(symbol2,
```

```
stac_bod[1])])
        display(Math("\\text{Tady je Hessova matice vyčíslená v bodě} \; %s: "
%(sym.latex(stac_bod))))
        display(dosazH)
        if is_positive(dosazH):
            display(Math("\\text{Funkce nabývá v bodě} \; %s \; \\text{ostré lokální minimum} \;
%s " %(sym.latex(stac_bod),sym.latex(f.subs({symbol1: stac_bod[0], symbol2: stac_bod[1]})))))
        elif is_negative(dosazH):
            display(Math("\\text{Funkce nabývá v bodě} \; %s \; \\text{ostré lokální maximum} \;
%s  " %(sym.latex(stac_bod),sym.latex(f.subs({symbol1: stac_bod[0], symbol2: stac_bod[1]})))))
        elif Matrix(dosazH).det() == 0:
            display(Math("\\text{Nelze rozhodnout zda podle kritéria zde bod} \; %s \;
\\text{nabývá lokálního extrému nebo je sedlem, protože determinant Hessovy matice se
rovná nule.}" %(sym.latex(stac_bod))))
        elif Matrix(dosazH).det() < 0:
          display(Math("\\text{Bod} \; %s \; \\text{je sedlovým bodem funkce.}"
%(sym.latex(stac_bod))))
        else:
            display(Math("\\text{Funkce v bodě} \; %s \; \\text{nenábývá lokální extrém.} "
%(sym.latex(stac_bod))))
    xsouradnice = []
    for bod in dfReal:
        xsouradnice.append(bod[0])
    minx = min(xsouradnice)
    maxx = max(xsouradnice)
    ysouradnice = []
    for bod in dfReal:
        ysouradnice.append(bod[1])
    miny = min(ysouradnice)
    maxy = max(ysouradnice)
    plot3d(f, (x, minx - 2,maxx + 2), (y, miny - 2, maxy + 2), axes=True, mesh=True)
```

(Vlastní zpracování v softwaru Jupyter Notebook; Kalus, Hřivňák,2001; Hamhalter, Tišer, 2005; Khan Academy 2021)

3.4.2 Testování

Kód byl důkladně otestován na následujících příkladech.

Určete lokální extrémy funkce:

a) $f(x,y) = x^2 + 2xy + 3y^2 + 5x + 2y,$

b) $f(x,y) = 2xy - 3x^2 - 2y^2 + x + y$

c) $f(x,y) = 2x^3 + xy^2 + 5x^2 + y^2$

d) $f(x,y) = x^3 + xy^2 - 2xy - 5x$

e) $f(x,y) = e^{\frac{x}{2}(x+y^2)}$

f) $f(x,y) = (x^2 + y^2)e^{-x^2-y^2}$

Výsledky:

a) $\min f\left(\frac{-13}{4},\frac{3}{4}\right) = -\frac{59}{8}$; b) $\max f\left(\frac{3}{10},\frac{2}{5}\right) = \frac{7}{20}$; c) $\max f\left(-\frac{5}{3},0\right) = \frac{125}{27}$, $\min f(0,0) = 0$; d) $\max f\left(-\sqrt{2},1\right) = 4\sqrt{2}$, $\min f\left(\sqrt{2},1\right) = -4\sqrt{2}$; e) $\min f(-2,0) = -\frac{2}{e}$; f) $\min f(0,0) = 0$ (Klaška, 2009)

3.4.3 Příklad

Nalezněte lokální extrémy funkce $f(x,y) = x^3 + y^3 + 9xy + 27$.

Řešení:

In [10]:

```
najdi_lok_ext_vice_prom(x**3 + y**3 + 9*x*y + 27,x,y)
```

Out [10]:

Hessova matice v obecném bodě je rovna matici: $\begin{bmatrix} 6x & 9 \\ 9 & 6y \end{bmatrix}$

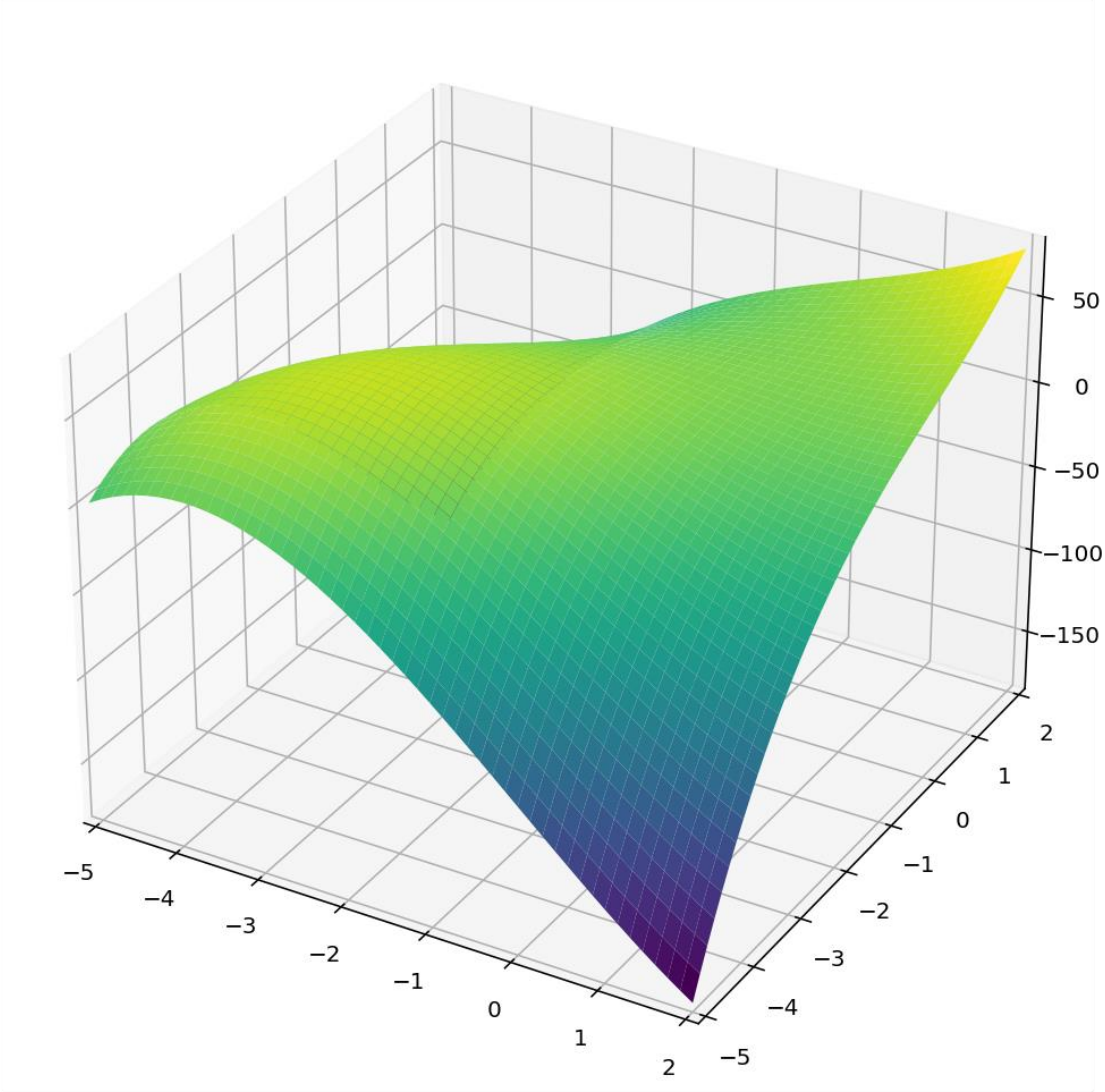
Tady je Hessova matice vyčíslená v bodě (-3, -3): $\begin{bmatrix} -18 & 9 \\ 9 & -18 \end{bmatrix}$

Funkce nabývá v bode (-3, -3) ostré lokální maximum 54

Tady je Hessova matice vyčíslená v bodě (0, 0): $\begin{bmatrix} 0 & 9 \\ 9 & 0 \end{bmatrix}$

Bod (0,0) je sedlovým bodem funkce.

Obrázek č. 5 – Graf funkce $f(x,y) = x^3 + y^3 + 9xy + 27$



Zdroj: Vlastní zpracování v programu Jupyter Notebook

3.5 Vázané extrémy funkcí více proměnných

Vázané extrémy jsou extrémy nějaké funkce vzhledem k předem zadaným podmínkám.

Definice

Řekněme, že funkce $f : \mathbb{R}^n \supseteq D_f \to \mathbb{R}$ má v bodě $A \in D_f$ lokální extrémy vázaný m podmínkami, $m < n$,

$$g_1(x_1, x_2, \cdots, x_n) = 0, g_2(x_1, x_2, \cdots, x_n) = 0, \cdots, g_m(x_1, x_2, \cdots, x_n) = 0,$$

Jestliže pro všechny body $X \in O(A) \subset D_f$, které vyhovují uvedeným podmínkám pak, funkce f má v bodě A:

- **Vázané lokální minimum** $\Leftrightarrow f(X) \geq f(A)$
- **Vázané lokální maximum** $\Leftrightarrow f(X) \leq f(A)$**.**

Na výpočet vázaných extrémů funkce využijeme Lagrangeovu metodu.

Lagrangeova Metoda

Mějme funkci $f : \mathbb{R}^n \supseteq D_f \rightarrow \mathbb{R}$, a nechť je dáno $m < n$ podmínek,

$g_j(x_1, x_2, \cdots, x_n) = 0.$

Jestliže má funkce Φ

$$\Phi(x_1, x_2, \cdots x_n, \lambda_1, \lambda_2, \cdots \lambda_n) = \mathrm{f}(x_1, x_2, \cdots x_n) + \sum_{j=1}^{m} \lambda_j g_j(x_1, x_2, \cdots x_n),$$

Kde $\lambda_j \in \mathbb{R}$, $j = 1,2, \cdots, m$, ve svém stacionárním bodě lokální extrém, má i funkce f v tomto bodě lokální extrém vázaný podmínknami $g_j(x_1, x_2, \cdots, x_n) = 0$.

Funkce Φ se nazývá Lagrangeova funkce a reálná čísla λ_j se nazývají Lagrangeovy multiplikátory

Extrémy Lagrangeovy funkce hledáme pomocí parciálních derivací všech proměnných. Tyto derivace položíme nule a získame soustavu n rovnic o n neznámých.

Soustavu vyřešíme a dostaneme body podezřelé z extrémů.

S podezřelími body naložíme stejně jako v případě u lokálních extrémů.

Dosadíme je do matice druhých parciálních derivací (tj. Hessovy matice) a podle Sylvestrova kritéria rozhodneme a jaký extrém se jedná.

Když je matice **pozitivně definitní** jedná se o **vázané lokální minimum** a když je matice **negativně definitní** jedná se o **vázané lokální maximum**.

(Kreml, 2007; Kalus, Hřivňák, 2001; Klaška, 2009)

3.5.1 Kód v Pythonu

Následující kód nalezne vázané extrémy zadané funkce.

In [12]:

```
def najdi_vaz_ext_vice_prom(f,g,symbol1,symbol2):
    """
    Funkce najdi_vaz_ext_vice_prom vypočítá vázané extrémy funkce více proměných

    Příklady
    ==========

    >>>najdi_vaz_ext_vice_prom(12*x + y - 3, - y - x**3+3,x,y)
    Mějme danou funkci f(x,y) = 12x + y - 3
    Omezenou funkcí g(x,y) = - x^3 - y + 3
    Lagrangeova funkce L = λ(− x^3 − y + 3) + 12x + y − 3
    Gradient L [− 3λx^2 + 12, 1 − λ]
    Soustava rovnic na výpočet stacionárních bodů [− 3λx^2 + 12, 1 − λ, − x^3 − y + 3]
    Stacionární body a Lambda [(−2, 11, 1), (2, −5, 1)]
    Matice druhých parciálních derivací :
    [-6λx 0]
    [0    0]
    Matice druhých parciálních derivací po dosazení :
    [12   0]
    [0    0]
    Funkce f(x,y) má vázané lokální minimum (x,y) = (−2,11)
    Matice druhých parciálních derivací po dosazení :
    [-12   0]
    [0     0]
    Funkce f(x,y) má vázané lokální maximum (x,y) = (2,-5)

    >>>najdi_vaz_ext_vice_prom(ln(x**2 + y**2), y - x - 3  ,x,y)
    Mějme danou funkci f(x,y) = log(x^2+y^2)
    Omezenou funkcí g(x,y) = - x + y - 3
    Lagrangeova funkce L = λ(- x + y - 3) + log(x^2+y^2)
    Gradient L [−λ + 2x/(x^2 + y^2), λ + 2y/(x^2 + y^2)]
    Soustava rovnic na výpočet stacionárních bodů [−λ + 2x/(x^2 + y^2), λ + 2y/(x^2 + y^2), − x +
y − 3]
    Stacionární body a Lambda [(−3/2, 3/2, −2/3)]
    Matice druhých parciálních derivací :
    [−4x^2/(x^2+y^2)^2 + 2x/(2+y^2)   −4xy/(x^2+y^2)^2]
    [−4xy/(x^2+y^2)^2   −4y^2/(x^2+y^2)^2 + 2x/(2+y^2)]
    Matice druhých parciálních derivací po dosazení :
    [0   4/9]
    [4/9   0]
    O existenci vázaného extrému nelze rozhodnout za pomocí Lagrangeovy funkce
    Funkce f(x,y) má vázané lokální minimum (x,y) = (−3/2,3/2)

    Parametry
    ==========
```

```
    f:
        - Funkce o dvou reálných proměnných; převážně f(x,y)
    g:
        - Omezující funkce. Je třeba poznamenat, že musí být vždy položena nule, proto je potřeba
ji příslušně upravit
          jestli tomu tak není.

    symbol1 :
        - objekt třídy sympy.core.symbol.Symbol
        - jedná se o objekt, pro který řešíme danou funkci
        - pro f(x,y) je to x

    symbol2:
        - objekt třídy sympy.core.symbol.Symbol
        - jedná se o objekt, pro který řešíme danou funkci
        - pro f(x,y) je to y

    """
    L = f + lam*g
    gradL = [sym.diff(L,c) for c in [symbol1,symbol2]]
    soustava = gradL + [g]
    stat_body = nonlinsolve(soustava, [symbol1, symbol2, lam])
    sbReal = []
    for koren in stat_body:
        if koren[0].is_real and koren[1].is_real and koren[2].is_real:
            sbReal.append(koren)
    while len(sbReal) == 0:
        stat_body = solve(soustava, [symbol1,symbol2, lam], rational = True)
        for koren in stat_body:
            if koren[0].is_real and koren[1].is_real and koren[2].is_real:
                sbReal.append(koren)
    dparder = derive_by_array(gradL,[x,y])
    display(Math("\\text{ Mějme danou funkci } f(x,y) = %s" %sym.latex(f)))
    display(Math("\\text{ Omezenou funkcí } g(x,y) = %s" %sym.latex(g)))
    display(Math("\\text{ Lagrangeova funkce } L = %s" %sym.latex(L)))
    display(Math("\\text{ Gradient L } %s" %sym.latex(gradL)))
    display(Math("\\text{ Soustava rovnic na výpočet stacionárních bodů } %s"
%sym.latex(soustava)))
    display(Math("\\text{ Stacionární body a Lambda }  %s" %sym.latex(sbReal)))
    display(Math("\\text{Matice druhých parciálních derivací :}"))
    display(dparder)
    for element in sbReal:
        if f.subs([(symbol1,element[0]), (symbol2,element[1])]).is_real:
            display(Math("\\text{Matice druhých parciálních derivací po dosazení :}"))
            dosparder =
dparder.subs([(symbol1,element[0]),(symbol2,element[1]),(lam,element[2])])
            display(dosparder)
            alpha = dosparder[0][0]
            pom = dosparder[1][1]
            diag_times = alpha * pom
            if diag_times<0:
                display(Math("\\text{Extrém pro Langrangeovu funkci neexistuje. } "))
```

```
        elif alpha > 0:
            display(Math("\\text{Funkce f(x,y) má vázané lokální minimum }\; (x,y)=(%s,%s) "
%(sym.latex(element[0]),sym.latex(element[1]))))
        elif alpha < 0:
            display(Math("\\text{Funkce f(x,y) má vázané lokální maximum } \; (x,y)=(%s,%s) "
%(sym.latex(element[0]),sym.latex(element[1]))))
        else:
            display(Math("\\text{O existenci vázaného extrému nelze rozhodnout za pomocí
Lagrangeovy funkce}"))
    else:
        pass
```

(Vlastní zpracování v sofwaru Jupyter Notebook ,Kalus, Hřivňák, 2001; Kreml, 2007; Isibalo,2017)

3.5.2 Testování

Kód byl důkladně otestován na následujících příkladech.

Určete vázané extrémy funkce:

a) $f(x,y) = 4x + 2y + 1,\ y = x^2 + x + \frac{1}{4}$

b) $f(x,y) = \sqrt{3}x - y + 2,\ x^2 + 2x + y^2 = 0$

c) $f(x,y) = xy - x + y - 1,\ x + y = 1$

d) $f(x,y) = x + y,\ x^2 + y^2 = 288$

e) $f(x,y) = ln(xy),\ x^2 + y^2 = 2$

f) $f(x,y) = 6x + 6y,\ x^3 + y^3 = 16$

Výsledky:

a) min $[-\frac{3}{2}, 1]$; b) max $[\frac{\sqrt{3}-2}{2}, -\frac{1}{2}]$, min $[-\frac{\sqrt{3}+2}{2}, \frac{1}{2}]$; c) nerozhodné; d) max [12, 12], min [-12, -12]; e) max [-1, -1],[1, 1]; f) max [2, 2] (Klaška, 2009; Hamhalter, Tišer, 2005)

3.5.3 Příklad

Nalezněte extrémy funkce $f(x,y) = x + y$ omezenou funkcí $1 = \frac{1}{x^2} + \frac{1}{y^2}$

Řešení:

In [13]:

```
najdi_vaz_ext_vice_prom(x+y,1/(x**2) + 1/(y**2) - 1,x,y)
```

Out [13]:

Mějme danou funkci $f(x,y) = x + y$

Omezenou funkcí $g(x,y) = -1 + \frac{1}{y^2} + \frac{1}{x^2}$

Lagrangeova funkce $L = \lambda(-1 + \frac{1}{y^2} + \frac{1}{x^2}) + x + y$

Gradient L $\left[-\frac{2\lambda}{x^3} + 1,\ -\frac{2\lambda}{y^3} + 1\right]$

Soustava rovnic na výpočet stacionárních bodů $\left[-\frac{2\lambda}{x^3} + 1,\ -\frac{2\lambda}{y^3} + 1, -1 + \frac{1}{y^2} + \frac{1}{x^2}\right]$

Stacionární body a Lambda $[(\sqrt{2},\sqrt{2},\sqrt{2}),(-\sqrt{2},-\sqrt{2},-\sqrt{2})]$

Matice druhých parciálních derivací $\begin{bmatrix} \frac{6\lambda}{x^4} & 0 \\ 0 & \frac{6\lambda}{y^4} \end{bmatrix}$

Matic druhých parciálních derivací po dosazení $\begin{bmatrix} \frac{3\sqrt{2}}{2} & 0 \\ 0 & \frac{3\sqrt{2}}{2} \end{bmatrix}$

Funkce $f(x,y)$ nabývá v bodě $(x,y) = \left(\sqrt{2},\sqrt{2}\right)$ nabývá vázané lokální minimum $2\sqrt{2}$

Matic druhých parciálních derivací po dosazení $\begin{bmatrix} -\frac{3\sqrt{2}}{2} & 0 \\ 0 & -\frac{3\sqrt{2}}{2} \end{bmatrix}$

Funkce $f(x,y)$ nabývá v bodě $(x,y) = \left(-\sqrt{2},-\sqrt{2}\right)$ nabývá vázané lokální minimum $-2\sqrt{2}$

Kód grafu

U předchozích příkladů jsou na utváření grafu využívány příkazy plot/plot3d, které pocházejí z knihovny Sympy. Bohužel Sympy nepodporuje složitější typy grafu, jako například zde, kde je pro ideální představu třeba na plochu definovanou funkcí $f(x,y) = x + y$ vyreslit křivku $g(x,y) = 0$.

Následující kód pochází z matematického softwaru Sagemath.

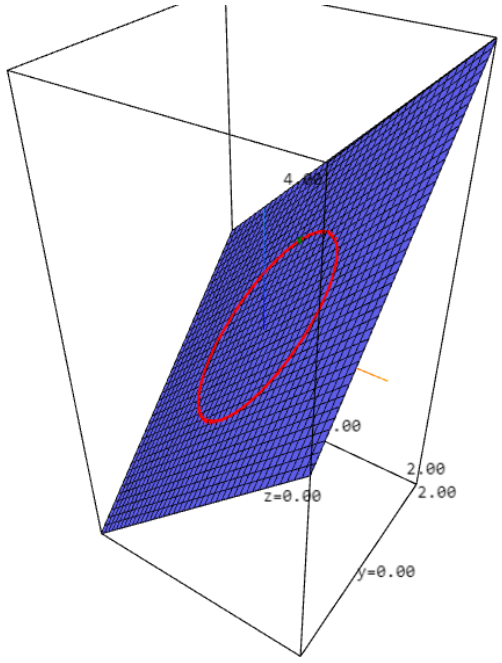
In [14]:

```
var("t,x,y")
f(x,y) = x+y
p1 = plot3d(f(x,y), (x, -2, 2), (y, -2, 2), axes=True, mesh=True)
p3 = parametric_plot((t, sqrt(-t**2/(-t**2 + 1)), f(t, sqrt(-t**2/(-t**2 + 1)))), (t, -10, 10),
color="red")
p4 = parametric_plot3d((cos(t), sin(t), f(cos(t), sin(t))), (t, 0, 2*pi), color="red", thickness=4)
(p1 + p2 + p4).show()
```

(Vlastní zpracování v programu Sagemath)

Out [14]:

Obrázek č. 6 – Graf funkce $f(x,y) = x + y$ s křivkou $-1 + \frac{1}{y^2} + \frac{1}{x^2} = 0$



Zdroj: Vlastní zpracování v programu Sagemath

3.6 Globální extrémy funkcí více proměnných

Definice

Řekněme, že funkce $f(x,y)$ má v bodě $A = [x_0, y_0]$ globální extrém na uzavřeném definičním oboru D_f, jestliže $\forall X \in D_f$ má funkce $f(x,y)$ v bodě A

- **Globální maximum** $\Leftrightarrow f(X) \leq f(A)$.
- **Globální minimum** $\Leftrightarrow f(X) \geq f(A)$.

V případě ostrých nerovností se jedná o ostré globální extrémy.

Globální extrémy hledáme na celém definičním oboru, včetně hraničních bodů.

Hledání globálních extrémů spočívá v nalezení extrémů s nejvyšší a nejnižší funkční hodnotou, proto využívá minulých postupů. Tj. hledáme lokální extrémy funkce na množině D_f, ze které vyloučíme hranici $g(x,y) = 0$ a určíme vázané extrémy funkce vzhledem k podmínce $g(x,y) = 0$, následovně porovnáme jejich funkční hodnoty a získáme nejvyšší (**globální maximum**) a nejnižší (**globální minimum**)

3.6.1 Kód v Pythonu

Následující kód na hledání globálních extrémů je rozčleněn na tři různé typy a dvě pomocné funkce.

Kód je takto rozčleněn z důvodu obvyklého zadávání úloh na globální extrémy funkcí dvou proměnných. Funkce, u které chceme zjistit globální extrémy, býva obvykle omezena třemi až čtyřmi body nebo křivkou.

Proto se tři kódy ve zpracování liší.

Kód hledající extrémy na úsečce

Tento kód vyhledá všechny extrémy, které se nacházejí na úsečce mezi dvěma body A[x,y] a B[x,y] vzhledem k funkci f(x,y), tudíž je potřeba na určení globálního extrému v případě, kdy je funkce ohraničena body.

In [15]:

```
def primka_ext(f,A,B):
    """
    Funkce nejprve nalezne přímku y = ax + b, která prochází body A(x,y) a B(x,y).
    Poté usoudí zda na této přímce leží extrém vzhledem k funkci f(x,y).
    Jestliže ano, pak tento extrém vrací na další spracování.

    Příklady
    ==========
    >>>primka_ext(3*x**2 + 2*y**2 + 3*x*y + 10*y, (1,-3),(5,-3))
    [(3/2, -3)]

    >>>primka_ext(3*x**2 + 2*y**2 + 3*x*y + 10*y, (1,-3),(1,-7))
    [(1, -13/4)]

    >>>primka_ext(3*x**2 + 2*y**2 + 3*x*y + 10*y, (5,-3),(1,-7))
    [(2.875, -5.125)]

    >>>primka_ext(-x**2 - y**2 + 2*y, (0,0),(-1,0))
    [(0,0)]

    >>>primka_ext(-x**2 - y**2 + 2*y, (-1,0),(-1,-1))
    [(-1,1)]

    >>>primka_ext(-x**2 - y**2 + 2*y, (-1,-1),(0,-1))
    [(0,-1)]

    >>>primka_ext(-x**2 - y**2 + 2*y, (0,-1),(0,0))
    [(0,1)]

    Parametry
    ==========
    f:
        - Funkce o dvou reálných proměnných; převážně f(x,y)
    A :
        - Bod
        - představuje vrchol útvaru, na kterém je omezená funkce f(x,y)
        - Souřadnice (x,y)
        - Spolu s B tvoří úsečku AB, která je stranou omezujícího útvaru, a na které později
hledáme extrém
    B:
        - Bod
        - představuje vrchol útvaru, na kterém je omezená funkce f(x,y)
        - Souřadnice (x,y)
        - Spolu s A tvoří úsečku AB, která je stranou omezujícího útvaru, a na které později
hledáme extrém

    """
    if A[0] == B[0]:
        lbp = []
```

```
        x_1 = A[0]
        f_1 = f.subs(x,x_1)
        df_1 = f_1.diff(y)
        y_1 = solve(df_1)
        for element in y_1:
            b_p = (x_1,element)
            ddf_1 = df_1.diff(y)
            if ddf_1 != 0:
                lbp.append(b_p)
        return lbp
    elif A[1] == B[1]:
        lbp = []
        y_1 = A[1]
        f_1 = f.subs(y,y_1)
        df_1 = f_1.diff(x)
        x_1 = solve(df_1)
        for element in x_1:
            b_p = (element,y_1)
            ddf_1 = df_1.diff(x)
            if ddf_1 != 0:
                lbp.append(b_p)
        return lbp
    else:
        lbp = []
        a = ((B[1] - A[1])) / (B[0] - A[0])
        b = (B[1] - (a * B[0]))
        y_1 = a*x + b
        f_1 = f.subs(y, y_1)
        df_1 = f_1.diff(x)
        x_1 = solve(df_1)
        for element in x_1:
            y_2 = y_1.subs(x, element)
            b_p = (element, y_2)
            ddf_1 = df_1.diff(x)
            if ddf_1 != 0:
                lbp.append(b_p)
        return lbp
```

(Vlastní zpracování v softwaru Jupyter Notebook; Isibalo, 2017; Onlineschool cz, 2017)

Kód Heronova vzorce

Kód za pomocí Heronova vzorce spočítá obsah trojúhelníku za předpokladu, že známe pouze souřadnice jeho vrcholů.

Při zpracování kódu na hledání globálních extrému je nezbytné si ujasnit, zda náš podezřelý bod leží v omezeném prostoru funkce podmínkou. V případě, že tento prostor tvoří trojúhelník či čtyřůhelník, toho dosáhneme tak, že využijeme Heronova vzorce na

spočítání obsahu trojúhelníku, kde jeden z vrcholů je náš podezřelý bod, a ostatní dva jsou vrcholy omezujícího trojúhelníku či čtyřúhelníku.

Obsahy spočítáme (tři nebo čtyři, záleží, o jaký útvar se jedná), sečteme a porovnáme s obsahem původního útvaru.

V případě, že se rovnají, pak podeřelý bod leží uvnitř vymezeného prostoru.

In [16]:

```
def troj_obsah(A,B,C):
    """
    Funkce vrací obsah trojúhelníku, u kterého známe pouze souřadnice vrcholů.
    Dosahuje toho za pomocí Heronova vzorce.

    Příklady
    ==========

    >>>troj_obsah((-12, 6),(4, 0),(-8, -6))
    84.0

    >>>troj_obsah((1, -3),(5, -3),(1,-7))
    8.0

    >>>troj_obsah((0, 0),(2, 0),(0, 1))
    1.0

    >>>troj_obsah((15, 5),(-4, 7),(1, 1))
    52.0

    Parametry
    ==========
    A:
        - Vrchol trojúhelníku ABC
        - Souřadnice (x,y)
    B:
        - Vrchol trojúhelníku ABC
        - Souřadnice (x,y)
    C:
        - Vrchol trojúhelníku ABC
        - Souřadnice (x,y)
    """
    a = sqrt((B[0] - A[0])**2 + (B[1] - A[1])**2)
    b = sqrt((C[0] - B[0])**2 + (C[1] - B[1])**2)
    c = sqrt((C[0] - A[0])**2 + (C[1] - A[1])**2)
    s = (a + b + c)/2
    obsah = sqrt(s * (s - a) * (s - b) * (s - c))
    if obsah.is_number and obsah.as_real_imag()[1]:
        return obsah
    else:
        return float(obsah)
```

(Vlastní zpracování v sofwaru Jupyter Notebook; Matematika polopatě 2022)

Kód na výpočet globálních extrémů funkce f(x,y) omezenou trojúhelníkem ABC

Následující kód využívá kódy předešlé k určení globálních extrémů funkce omezené trojúhelníkem.

In [17]:

```
def glob_ext_trojuhelnik(f, A, B, C):
    """
    Funkce vypočítá globální extrémy funkce f(x,y), omezenou trojúhelníkem ABC, kde
A(x,y),B(x,y) a C(x,y).

    Příklady
    ==========
    >>>glob_ext_trojuhelnik(-x**2 - y**2 + 2*y, (1,4),(-2,1),(0,-1))
    Funkce nabývá minima f(x,y) = -9 v bodě(1, 4)
    Funkce nabývá maxima f(x,y) = 1 v bodě(0, 1)

    >>>glob_ext_trojuhelnik(3*x**2 + 2*y**2 + 3*x*y + 10*y, (1,-3),(5,-3),(1,-7))
    Funkce nabývá minima f(x,y) = −20 v bodě (2, −4)
    Funkce nabývá maxima f(x,y) = 18 v bodě (5, −3)

    >>>glob_ext_trojuhelnik(x**2 - 2*y**2 + 4*x*y - 6*x -1, (0,0),(3,0),(0,3))
    Funkce nabývá minima f(x,y) = -19 v bodě (0, 3)
    Funkce nabývá maxima f(x,y) = -1 v bodě (0, 0)

    Parametry
    ==========
    f:
        - Funkce o dvou reálných proměnných; převážně f(x,y)
    A:
        - Souřadnice (x,y)
        - Vrchol trojúhelníku ABC
    B:
        - Souřadnice (x,y)
        - Vrchol trojúhelníku ABC
    C:
        - Souřadnice (x,y)
        - Vrchol trojúhelníku ABC
    """
    podezrele_body = [A,B,C]
    a_b = primka_ext(f,A,B)
    if a_b is None :
        pass
    elif len(a_b) > 1:
        for element in a_b:
            podezrele_body.append(element)
    elif len(a_b) == 1:
        podezrele_body.append(a_b[0])
    a_c = primka_ext(f,A,C)
```

```
    if a_c is None :
        pass
    elif len(a_c) > 1:
        for element in a_c:
            podezrele_body.append(element)
    elif len(a_c) == 1:
        podezrele_body.append(a_c[0])
    b_c = primka_ext(f,B,C)
    if b_c is None :
        pass
    elif len(b_c) > 1:
        for element in b_c:
            podezrele_body.append(element)
    elif len(b_c) == 1:
        podezrele_body.append(b_c[0])
    gradf = derive_by_array(f, [x,y])
    df = nonlinsolve(gradf, [x,y])
    dfReal = []
    for koren in df:
        if koren[0].is_real and koren[1].is_real:
            dfReal.append(koren)
    if len(dfReal) != 0:
        for stac_bod in dfReal:
            dosazH = Hess_Mat(f, x, y).subs([(x, stac_bod[0]),(y,  stac_bod[1])])
            if is_positive(dosazH) or is_negative(dosazH):
                obsah_1 = troj_obsah(A, B, stac_bod)
                obsah_2 = troj_obsah(A, stac_bod, C)
                obsah_3 = troj_obsah(stac_bod, B, C)
                obsahA = troj_obsah(A, B, C)
                obsahB = obsah_1 + obsah_2 + obsah_3
                if obsahA == obsahB:
                    podezrele_body.append(stac_bod)
            else:
                pass
    hodnoty = []
    for element in podezrele_body:
        hodnota = f.subs([(x, element[0]),(y, element[1])])
        hodnoty.append(hodnota)
    minim = min(hodnoty)
    maxim = max(hodnoty)
    display(Math("\\text{Funkce nabývá minima} \; f(x,y) =  %s \; \\text{v bodě} \; %s"
%(sym.latex(minim),sym.latex(podezrele_body[hodnoty.index(minim)]))))
    display(Math("\\text{Funkce nabývá maxima} \; f(x,y) =  %s \; \\text{v bodě} \; %s"
%(sym.latex(maxim),sym.latex(podezrele_body[hodnoty.index(maxim)]))))
```

(Vlastní zpracování v sofwaru Jupyter Notebook; Isibalo, 2017; Onlineschool cz, 2017)

Kód na výpočet globálních extrémů funkce f(x,y) omezenou čtyřúhelníkem ABCD

Tento kód využívá kódy předešlé k určení globálních extrémů funkce omezené čtyřúhelníkem.

In [18]:

```
def glob_ext_ctyruhelnik(f, A, B, C, D):
    """
    Funkce vypočítá globální extrémy funkce f(x,y), omezenou čtyřúhelníkem ABCD, kde
A(x,y),B(x,y),C(x,y) a D(x,y).

    Příklady
    ==========
    >>>glob_ext_ctyruhelnik((x-y)**2 + x**2, (2,0),(0,2),(-2,0),(0,-2))
    Funkce nabývá minima f(x,y) = 0 v bodě (0, 0)
    Funkce nabývá maxima f(x,y) = 8 v bodě (2, 0)

    >>>glob_ext_ctyruhelnik(-x**2 - y**2 + 2*y, (0,0),(-1,0),(-1,-1),(0,-1))
    Funkce nabývá minima f(x,y) = -4 v bodě (-1, -1)
    Funkce nabývá maxima f(x,y) = 0 v bodě (0, 0)

    Parametry
    ==========
    f:
        - Funkce o dvou reálných proměnných; převážně f(x,y)
    A:
        - Souřadnice (x,y)
        - Vrchol čtyřúhelníku ABCD
    B:
        - Souřadnice (x,y)
        - Vrchol čtyřúhelníku ABCD
    C:
        - Souřadnice (x,y)
        - Vrchol čtyřúhelníku ABCD
    D:
        - Souřadnice (x,y)
        - Vrchol čtyřúhelníku ABCD

    """

    podezrele_body = []
    a_b = primka_ext(f,A,B)
    if a_b is None :
        pass
    elif len(a_b) > 1:
        for element in a_b:
            podezrele_body.append(element)
    elif len(a_b) == 1:
        podezrele_body.append(a_b[0])
```

```
b_c = primka_ext(f,B,C)
if b_c is None :
    pass
elif len(b_c) > 1:
    for element in b_c:
        podezrele_body.append(element)
elif len(b_c) == 1:
    podezrele_body.append(b_c[0])
c_d = primka_ext(f,C,D)
if c_d is None :
    pass
elif len(c_d) > 1:
    for element in c_d:
        podezrele_body.append(element)
elif len(c_d) == 1:
    podezrele_body.append(c_d[0])
d_a = primka_ext(f,D,A)
if d_a is None :
    pass
elif len(d_a) > 1:
    for element in d_a:
        podezrele_body.append(element)
elif len(d_a) == 1:
    podezrele_body.append(d_a[0])
gradf = derive_by_array(f, [x,y])
df = nonlinsolve(gradf, [x,y])
dfReal = []
for koren in df:
    if koren[0].is_real and koren[1].is_real:
        dfReal.append(koren)
if len(dfReal) != 0:
    for stac_bod in dfReal:
        dosazH = Hess_Mat(f, x, y).subs([(x, stac_bod[0]),(y,  stac_bod[1])])
        if is_positive(dosazH) or is_negative(dosazH):
            podezrele_body.append(stac_bod)
        else:
            pass
pod_body = [A,B,C,D]
for element in podezrele_body:
    obsah_1 = troj_obsah(A, B, element)
    obsah_2 = troj_obsah(B, C, element)
    obsah_3 = troj_obsah(C, D, element)
    obsah_4 = troj_obsah(D, A, element)
    obsahA = sqrt((B[0] - A[0])**2 + (B[1] - A[1])**2) * sqrt((C[0] - B[0])**2 + (C[1] - B[1])**2)
    obsahB = obsah_1 + obsah_2 + obsah_3 + obsah_4
    if obsahA == obsahB:
        pod_body.append(element)
hodnoty = []
for element in pod_body:
    hodnota = f.subs([(x, element[0]),(y, element[1])])
    hodnoty.append(hodnota)
minim = min(hodnoty)
```

```
    maxim = max(hodnoty)
    display(Math("\\text{Funkce nabývá minima} \; f(x,y) =  %s \; \\text{v bodě} \; %s"
%(sym.latex(minim),sym.latex(pod_body[hodnoty.index(minim)])))))
    display(Math("\\text{Funkce nabývá maxima} \; f(x,y) =  %s \; \\text{v bodě} \; %s"
%(sym.latex(maxim),sym.latex(pod_body[hodnoty.index(maxim)])))))
```

(Vlastní zpracování v sofwaru Jupyter Notebook; Isibalo, 2017; Onlineschool cz, 2017)

Kód na výpočet globálních extrémů funkce f(x,y) omezenou křivkou g

Následující kód využívá kódy předešlé k určení globálních extrémů funkce omezené křivkou.

In [19]:

```
def glob_ext_k(f, g):
    """
    Funkce vypočítá globální extrémy funkce f(x,y), omezené křivkou g. Převážně elipsou či
kružnicí.

    Příklady
    ==========
    >>>glob_ext_k(9*x**2 -36*x + 16*y**2 -64*y, 9*x**2 + 16*y**2 - 144)
    Funkce nabývá minima f(x,y) = -100 v bodě (2, 2)
    Funkce nabývá maxima f(x,y) = 384 v bodě (-12/5, -12/5)

    >>>glob_ext_k(4*y, x**2 + y**2 - 1)
    Funkce nabývá minima f(x,y) = -4 v bodě (0, -1)
    Funkce nabývá maxima f(x,y) = 4 v bodě (0, 1)

    >>>glob_ext_k(3*x + 4*y + 1, (x-3)**2 + (y-1)**2 -1)
    Funkce nabývá minima f(x,y) = 9 v bodě (12/5, 1/5)
    Funkce nabývá maxima f(x,y) = 19 v bodě (18/5, 9/5)

    Parametry
    ==========
    f:
        - Funkce o dvou reálných proměnných; převážně f(x,y)
    g:
        - Funkce o dvou reálných proměnných; převážně g(x,y)
        - Jedná se o implicitně zadanou funkci, proto je důležité funkci zadat ve tvaru g(x,y) = 0

    """
    podezrele_body = []
    gradf = derive_by_array(f, [x,y])
    df = nonlinsolve(gradf, [x,y])
    dfReal = []
    for koren in df:
        if koren[0].is_real and koren[1].is_real:
            dfReal.append(koren)
    if len(dfReal) != 0:
```

```
    for stac_bod in dfReal:
        dosazH = Hess_Mat(f, x, y).subs([(x, stac_bod[0]),(y,  stac_bod[1])])
        if is_positive(dosazH) or is_negative(dosazH):
            dosaz_g = g.subs([(x, stac_bod[0]), (y, stac_bod[1])])
            if dosaz_g <= 0:
                podezrele_body.append(stac_bod)
            else:
                pass
        else:
            pass
L = f + lam*g
gradL = [sym.diff(L,c) for c in [x,y]]
soustava = gradL + [g]
stat_body = nonlinsolve(soustava, [x, y, lam])
sbReal = []
for koren in stat_body:
    if koren[0].is_real and koren[1].is_real and koren[2].is_real:
        sbReal.append(koren)
while len(sbReal) == 0:
    stat_body = solve(soustava, [x,y, lam], rational = True)
    for koren in stat_body:
        if koren[0].is_real and koren[1].is_real and koren[2].is_real:
            sbReal.append(koren)
dparder = derive_by_array(gradL,[x,y])
for element in sbReal:
    if f.subs([(x,element[0]), (y,element[1])]).is_real:
        dosparder = dparder.subs([(x,element[0]),(y,element[1]),(lam,element[2])])
        alpha = dosparder[0][0]
        pom = dosparder[1][1]
        diag_times = alpha * pom
        if diag_times<0:
            pass
        elif alpha != 0:
            l = (element[0],element[1])
            podezrele_body.append(l)
    else:
        pass
hodnoty = []
for element in podezrele_body:
    hodnota = f.subs([(x, element[0]),(y, element[1])])
    hodnoty.append(hodnota)
minim = min(hodnoty)
maxim = max(hodnoty)
display(Math("\\text{Funkce nabývá minima} \; f(x,y) =  %s \; \\text{v bodě} \; %s"
%(sym.latex(minim),sym.latex(podezrele_body[hodnoty.index(minim)]))))
display(Math("\\text{Funkce nabývá maxima} \; f(x,y) =  %s \; \\text{v bodě} \; %s"
%(sym.latex(maxim),sym.latex(podezrele_body[hodnoty.index(maxim)]))))
```

(Vlastní zpracování v sofwaru Jupyter Notebook; Isibalo, 2017; Onlineschool cz, 2017)

3.6.2 Testování

Kód byl důkladně otestován na následujících příkladech.

Určete globální extrémy funkce:

a) $f(x,y) = x^2 - y;\ A(1,1)\ B(1,3)\ C(3,3)\ D(3,1)$

b) $f(x,y) = x^2 + y^2; A(0,0)\ B(2,0)\ C(0,1)$

c) $f(x,y) = 3x + 4y + 1, (x-3)^2 + (y-1)^2 \leq 1$

d) $f(x,y) = 5x - 3y + 1;\ A(1,4)B(-2,1)C(0,-1)$

e) $f(x,y) = -x^2 - y^2 + 2y;\ A(0,0)B(-1,0)C(-1,-1)D(0,-1)$

f) $f(x,y) = 2x + 4y + 1, x^2 + 4y^2 \leq 1$

Výsledky:

a) min $[1,3]$, max [3,1] ; b) min [0,0], max [2,0] ; c) min $[\frac{12}{5},\frac{1}{5}]$, max $[\frac{18}{5},\frac{9}{5}]$; d) min [-2,1], max [0,-1] ; e) min[-1,-1], max[0,0] ; f) min $[-\frac{\sqrt{2}}{2},-\frac{\sqrt{2}}{4}]$, max $[\frac{\sqrt{2}}{2},\frac{\sqrt{2}}{4}]$ (Morávková, 2014)

3.6.3 Příklady

1) Určete globální extrémy funkce $f(x,y) = 3x^2 + 2y^2 + 3xy + 10y$ omezené trojúhelníkem s vrcholy A(1,-3) B(5,-3) C(1,-7).

Řešení:

In [20]:

```
glob_ext_trojuhelnik(3*x**2 + 2*y**2 + 3*x*y + 10*y, (1,-3),(5,-3),(1,-7))
```

Out [20]:

Funkce nabývá minima $f(x,y) = -20$ v bodě $(2,-4)$

Funkce nabývá maxima $f(x,y) = 18$ v bodě $(5,-3)$

Kód grafu pomocí Matplotlib

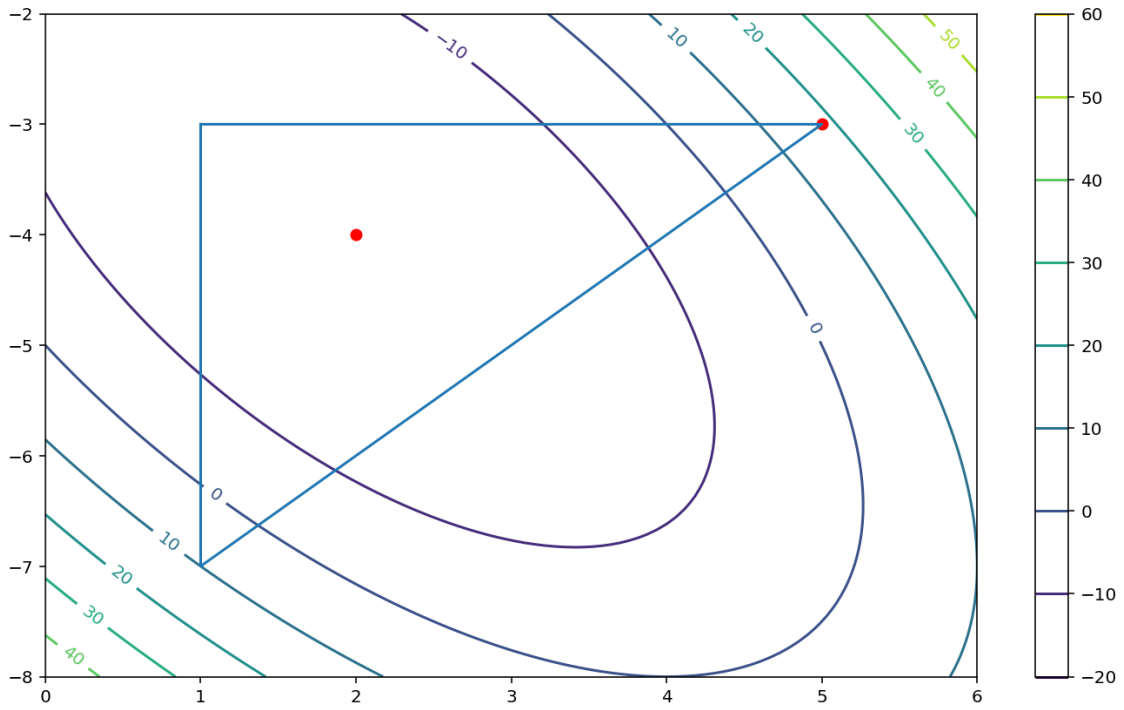
In [21]:

```
def fun(x, y):
    return 3*x**2 + 2*y**2 + 3*x*y + 10*y
xlist = np.linspace(0.0, 6.0, 100)
ylist = np.linspace(-8.0, -2.0, 100)
X, Y = np.meshgrid(xlist, ylist)
Z = fun(X,Y)
x = ([1,5,1,1])
y = ([-3,-3,-7, -3])
mx = [(2,5)]
my = [(-4,-3)]
fig,ax=plt.subplots(1,1)
cp = ax.contour(X, Y, Z)
fig.colorbar(cp)
plt.clabel(cp, inline = True, fontsize = 10)
plt.scatter(mx,my, color="red")
plt.plot(np.asarray(x), np.asarray(y))
plt.show()
```

(Vlastní zpracování v softwaru Jupyter Notebook)

Out [21]:

Obrázek č. 7 – Graf funkce $f(x,y) = 3x^2 + 2y^2 + 3xy + 10y$ v konturovém tvaru spolu s trojúhelníkem ABC a body reprezentující globální extrémy v rámci trojúhelníku ABC



Zdroj: Vlastní zpracování v programu Jupyter Notebook

2) Určete globální extrémy funkce $f(x,y) = x^3 + x^2y - 4y$ omezené obdélníkem s vrcholy A(1,4) B(1,-4) C(-1,-4) D(-1,4)

Řešení:

In [22]:

```
glob_ext_ctyruhelnik(x**3 + (x**2)*y - 4*y,(1,4),(1,-4),(-1,-4),(-1,4))
```

Out [22]:

Funkce nabývá minima $f(x,y) = -16$ v bodě $(0,4)$

Funkce nabývá maxima $f(x,y) = 16$ v bodě $(0,-4)$

Kód grafu pomocí Matplotlib

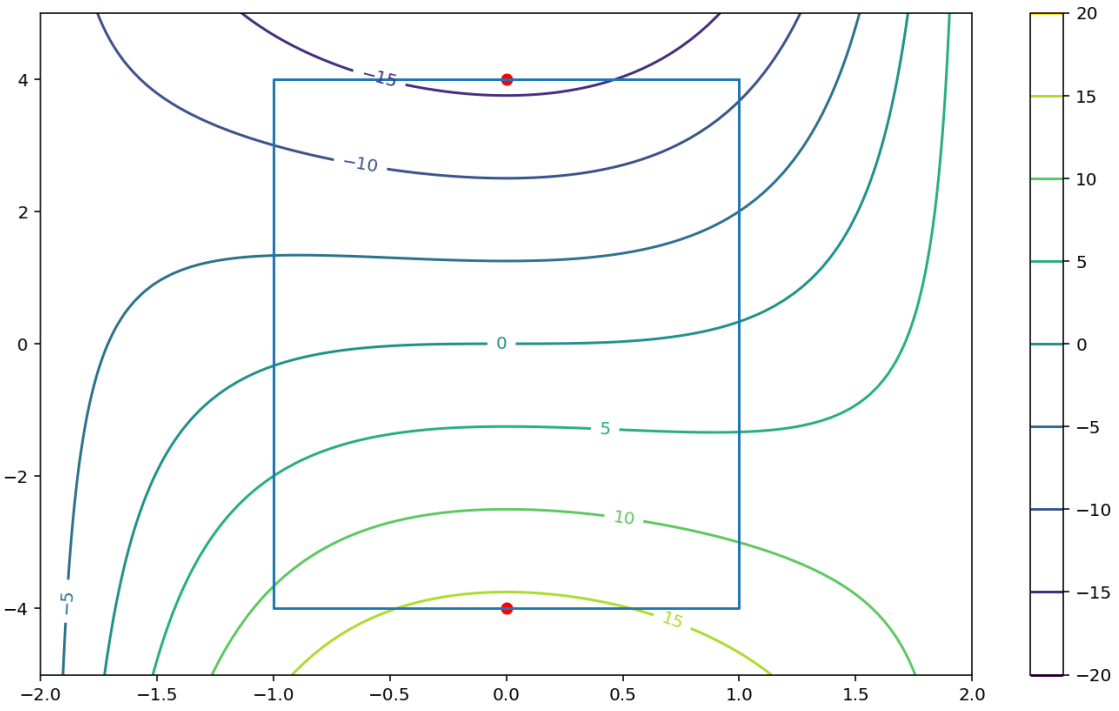
In [23]:

```
def fun(x, y):
    return x**3 + (x**2)*y - 4*y
xlist = np.linspace(-2.0, 2.0, 100)
ylist = np.linspace(-5.0, 5.0, 100)
X, Y = np.meshgrid(xlist, ylist)
Z = fun(X,Y)
x = ([1,1,-1,-1,1])
y = ([4,-4,-4,4,4])
mx = [(0,0)]
my = [(4,-4)]
fig,ax=plt.subplots(1,1)
cp = ax.contour(X, Y, Z)
fig.colorbar(cp)
plt.clabel(cp, inline = True, fontsize = 10)
plt.scatter(mx,my, color="red")
plt.plot(np.asarray(x), np.asarray(y))
plt.show()
```

(Vlastní zpracování v softwaru Jupyter Notebook)

Out [23]:

Obrázek č. 8 – Graf funkce $f(x,y) = x^3 + x^2y - 4y$ v konturovém tvaru spolu s obdélníkem ABCD a body reprezentující globální extrémy



Zdroj: Vlastní zpracování v programu Jupyter Notebook

3) Určete globální extrémy funkce $f(x,y) = 9x^2 - 36x + 16y^2 - 64y$ omezené elipsou $9x^2 + 16y^2 \leq 144$

Řešení:

In [24]:

```
glob_ext_k(9*x**2 - 36*x + 16*y**2 - 64*y, 9*x**2 + 16*y**2 - 144)
```

Out [24]:

Funkce nabývá minima $f(x,y) = -100$ v bodě $(2,2)$

Funkce nabývá maxima $f(x,y) = 384$ v bodě $(-\frac{12}{5}, -\frac{12}{5})$

Kód grafu pomocí Matplotlib

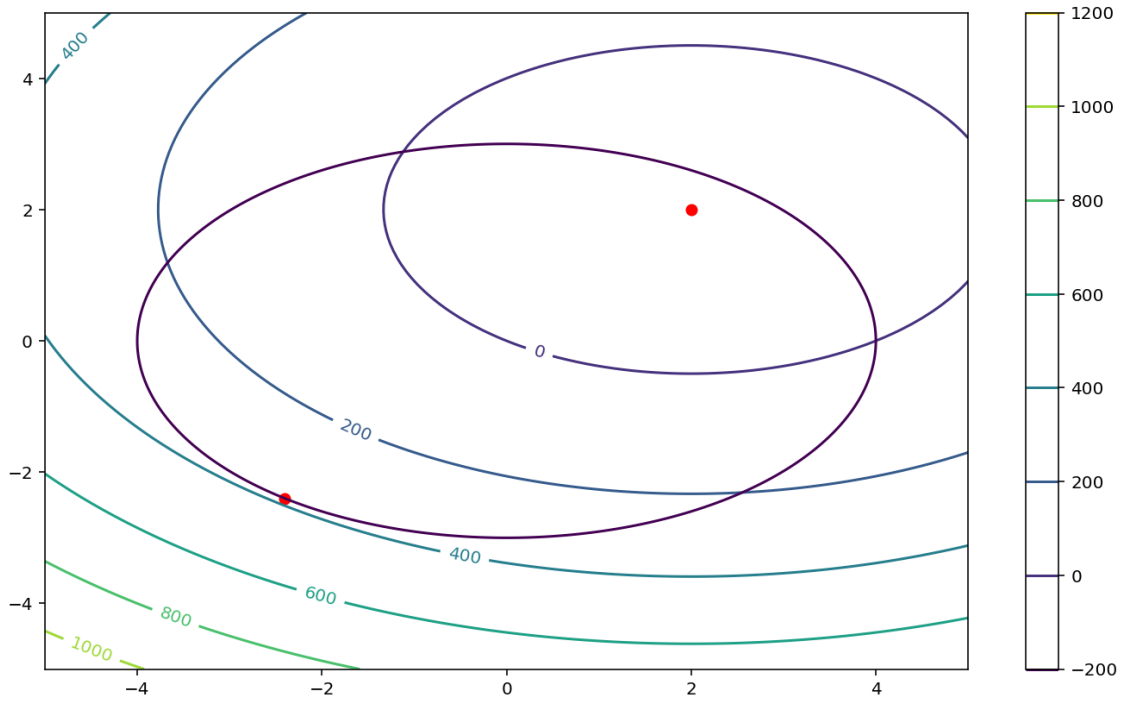
In [25]:

```
def f(x, y):
    return 9*x**2 - 36*x + 16*y**2 - 64*y
xlist = np.linspace(-5.0, 5.0, 100)
ylist = np.linspace(-5.0, 5.0, 100)
xx, yy = np.meshgrid(xlist, ylist)
Z = f(xx,yy)
xrange = np.arange(-5.0, 5.0, 0.025)
yrange = np.arange(-5.0, 5.0, 0.025)
X, Y = np.meshgrid(xrange,yrange)
F = 9*X**2
G = 144 - 16*Y**2
mx = [(2,-12/5)]
my = [(2,-12/5)]
fig,ax=plt.subplots(1,1)
cp = ax.contour(xx, yy, Z)
fig.colorbar(cp)
plt.clabel(cp, inline = True, fontsize = 10)
plt.scatter(mx,my, color="red")
plt.contour(X, Y, (F - G), [0])
plt.show()
```

(Vlastní zpracování v softwaru Jupyter Notebook)

Out [25]:

Obrázek č. 9 – Graf funce $f(x,y) = 9x^2 - 36x + 16y^2 - 64y$ v konturovém tvaru spolu s elipsou ve tvaru $9x^2 + 16y^2 \leq 144$ a body reprezentující globální extrémy



Zdroj: Vlastní zpracování v programu Jupyter Notebook

3.7 Gradientní sestup

Gradientní sestup je optimalizační algoritmus, který je postaven na konvexní funkce a postupně upravuje její parametry v jednotlivých iteracích, dokud nezminimalizuje danou funkci do jejího lokálního minima.

Začne se definováním prvotních paramatrů a odtud gradientní sestup vyžitím matematického počtu určí minimum dané funkce.

Gradient je sklon, který měří, jak moc se výsledek dané funkce změní, když se změní její vstupní hodnoty. Matematicky řečeno, představuje parciální derivace funkce s ohledem na její vstupní hodnoty.

Gradient:

$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{bmatrix}$$

Nejjednodušeji lze gradient nazvat směrem růstu dané funkce.

Následující funkce reprezentuje gradientní sestup.

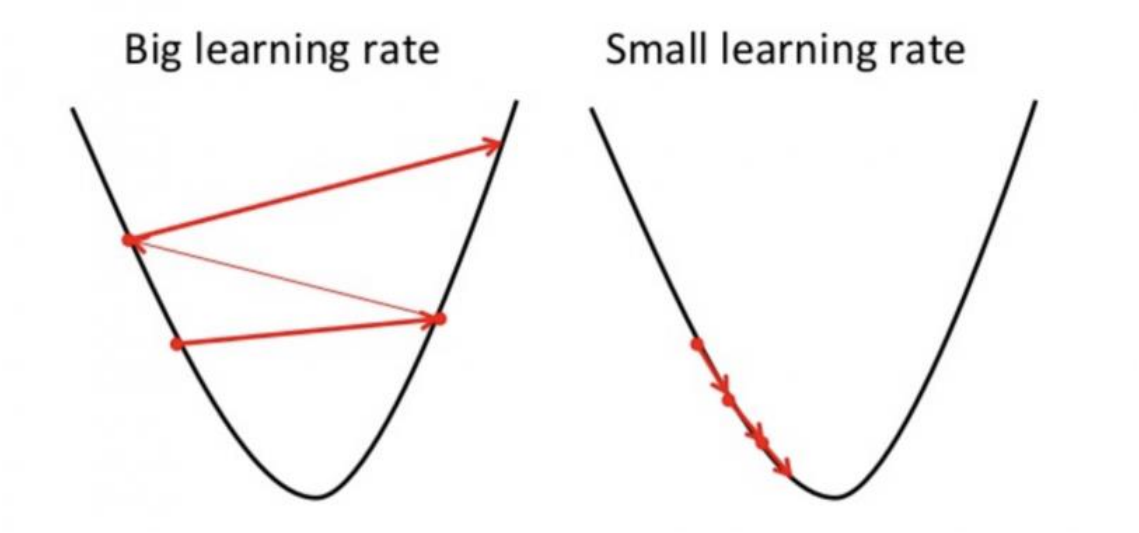
$$b = a - \eta \, \nabla f(a)$$

Kde b je hodnota, kterou získáme provedením iterace, a je naše součastná hodnota po předešlé iteraci (nebo zvolená počáteční hodnota), η je délka kroku a $\nabla f(a)$ představuje směr nejrychlejšího sestupu neboli gradient funkce $f(a)$.

Délka kroku určí rychlost gradientního sestupu.

Aby gradientní sestup dosáhl lokálního minima, musíme délce kroku zadat vhodnout hodnotu, to znamená, že nemůže být příliš vysoká nebo nízká. Její přesné určení je důležité, protože když jsou kroky moc velké, gradientní sestup nemusí dosáhnout minima, tak se sestup vrací tam a zpátky mezi rameny konvexní funkce. Jestliže zvolíme délu kroku nízkou, tak sice minima dosáhne, ale bude to nějakou chvilku trvat.

Obrázek č. 10 – Délka kroku gradientního sestupu



Zdroj: https://builtin.com/data-science/gradient-descent

Proto by hodnota délky kroku nemělá být moc vysoká, ani moc nízká.

(Donges, 2021; Bachman, 2007)

3.7.1 Kód v Pythonu pro jednu proměnnou

Následující kód využívá gradientního sestupu k dosažení minima funkce jedné proměnné.

In [26]:

```
def grad_sestup_1(f, x0, η):
    """
    Funkce grad_sestup_1 představuje metodu gradientního sestupu pro hledání minima funkce jedné proměnné.

    Příklady
    ==========

    Pří využívání funkce grad_sestup_1 nejvíce záleží no volbě parametru x0

    Správně zvolený parametr vypadá takto:

    >>>grad_sestup_1(x**3 - 2*x**2 -4*x + 5,0,0.1)
    Iterace 1 Hodnota x:  0.400000000000000
    Iterace 2 Hodnota x:  0.912000000000000
    .
    .
    .
```

Iterace 12 Hodnota x: 1.99999920213583
Iterace 13 Hodnota x: 1.99999984042697
Funkce nabývá lokálního minima pro x = 2.0

V případě kdy se přímo trefíme do minima:

>>>grad_sestup_1(x**3 - 2*x**2 -4*x + 5,2,0.1)
Iterace 1 Hodnota x: 2
Narazili jste na lokální minimum x = 2

Zjistili jsme, že lokální minimum se nachází v x = 2, musíme ale zjistit zdali exitují jiná minima na funkci f(x).
Docílí me toho postupným měněním x0.

Od první volby x0 nejprve postupně navyšujeme. Postupným navyšováním zjistíme, že nám funkce vrátí jiný výsledek
pro x0 = 6

>>>grad_sestup_1(x**3 - 2*x**2 -4*x + 5,6,0.1)
Iterace 1 Hodnota x: -2.00000000000000
Iterace 2 Hodnota x: -3.60000000000000
Iterace 3 Hodnota x: -8.52800000000000
Iterace 4 Hodnota x: -33.3572352000000
Iterace 5 Hodnota x: -380.111671336436
Iterace 6 Hodnota x: -43877.2211457246
Error. Blížíme se k nekonečnu. Prosím změnte x0

Dalším navyšováním zjistíme, že se výsledek od šesti a více nemění. Začneme tedy x0 od původního snižovat.

>>>grad_sestup_1(x**3 - 2*x**2 -4*x + 5,-1,0.1)
Iterace 1 Hodnota x: -1.30000000000000
Iterace 2 Hodnota x: -1.92700000000000
Iterace 3 Hodnota x: -3.41179870000000
Iterace 4 Hodnota x: -7.86862929079651
Iterace 5 Hodnota x: -29.1906790819093
Iterace 6 Hodnota x: -296.095674293579
Iterace 7 Hodnota x: -26715.9284446218
Error. Blížíme se k nekonečnu. Prosím změnte x0

A ihned zjistíme, že při zvolení mínus jedna a nižších čísel se blížíme k nekonečnu.
Lze tedy usoudit, že funkce x^3 - 2x^2 + 5 má pouze jedno minimum pro x = 2

Při překročení 10000 iterací se funkce zastaví a dá nám radu:

>>>grad_sestup_1((x**4 + 4*x**3 - 8*x**2)/4,-3,0.1)
Iterace 1 Hodnota x: -4.20000000000000
Iterace 2 Hodnota x: -3.76320000000000
Iterace 3 Hodnota x: -4.18766102200320
.
.

```
.
Iterace 9999 Hodnota x:  -4.00835149685264
Iterace 10000 Hodnota x:  -3.99158567214804
Iterace 10001 Hodnota x:  -4.00835066660429
V případě, že funkce překročí 10000 iterací, tak se sama zastaví. Pokud se x0 stále zvyšuje nebo snižuje, pak se blížíme k nekonečnu a je třeba zvolit jinné x0.
Jestli tomu tak není, potom zadejte x0 jako číslo, okolo kterého se pohybuje.

Když se zachováme podle rady:

>>>grad_sestup_1((x**4 + 4*x**3 - 8*x**2)/4,-4,0.1)
Iterace 1 Hodnota x:  -4
Narazili jste na lokální minimum x = −4

Samozřejmě u určitých funkcí nelze volit x0 pouze jako celé číslo:

>>>grad_sestup_1( sin(x)*cos(x),1,0.1)
Iterace 1 Hodnota x:  -0.1*cos(1)**2 + 0.1*sin(1)**2 + 1
Error. Prosím změnte x0

>>>grad_sestup_1( sin(x)*cos(x),0.5,0.1)
Iterace 1 Hodnota x:  0.445969769413186
Iterace 2 Hodnota x:  0.383179401130255
.
.
.
Iterace 61 Hodnota x:  -0.785392535181723
Iterace 62 Hodnota x:  -0.785393660824868
Iterace 63 Hodnota x:  -0.785394561339384
Funkce nabývá lokálního minima pro x = − 0.79

Poslední x se vždy zaokrouhluje, ale iterace 63 je téměř totožná s -π/4, což by bylo vtomto případě minimum.

Parametry
==========
f:
   - Funkce reálné proměnné
x0:
   - vstupní hodnota x
   - číslo
   - Počáteční bod, od kterého začínáme sestupovat.
η:
   - rozhoduje o rychlosti sestupu
   - Takzvanná "délka" kroku.
"""
fp = f
xu = x0
df = sym.diff(f,x)
ddf = df.diff(x)
q = 0
vk = 1
```

```
    global seznam_x
    global seznam_y
    seznam_x = []
    seznam_y = []
    while vk > 0.000001:
        xo = x0
        x0 = x0 - η * df.subs(x, xo)
        vk = (abs(x0-xo))
        q = q + 1
        print("Iterace",q,"Hodnota x: ",x0)
        seznam_x.append(x0)
        if q > 10000:
            break
        elif vk > 10000:
            break
        elif isinstance(x0, sym.core.add.Add):
            break
        else:
            continue
    for bod in seznam_x:
        y = f.subs(x,bod)
        seznam_y.append(y)
    if q > 10000:
        display(Math("\\text{V případě, že funkce překročí 10000 iterací, tak se sama zastaví.
Pokud se x0 stále zvyšuje nebo snižuje, pak se blížíme k nekonečnu a je třeba zvolit jinné x0.}"))
        display(Math("\\text{Jestli tomu tak není, potom zadejte x0 jako číslo, okolo kterého se
pohybuje.}"))
    elif q > 2000:
        minx = round(x0, 1)
        display(Math("\\text{Funkce nabývá lokálního minima pro } \; x \; = %s"
%sym.latex(minx)))
    elif isinstance(x0, sym.core.add.Add):
        display(Math("\\text{Error. Prosím změnte x0}"))
    elif q == 1:
        if ddf.subs(x,xu) > 0:
            display(Math("\\text{Narazili jste na lokální minimum } \; x \; = %s" %sym.latex(xu)))
        else:
            display(Math("\\text{Narazili jste na lokální maximum nebo inflexní bod. Zkuste změnit
x0}"))
    elif vk > 10000:
        display(Math("\\text{Error. Blížíme se k nekonečnu. Prosím změnte x0}"))
    else:
        minx = round(x0, 2)
        if ddf.subs(x,minx) > 0:
            display(Math("\\text{ Funkce nabývá lokálního minima pro } \; x \; = %s"
%sym.latex(minx)))
        else:
            display(Math("\\text{Narazili jste na lokální maximum nebo inflexní bod. Zkuste změnit
x0}"))
```

(Vlastní zpracování v software Jupyter Notebook; Donges, 2021; Crypto1, 2020; StatQuestWithJoshStarmer, 2019)

3.7.2 Testování pro jednu proměnnou

Kód byl důkladně otestován na následujících příkladech.

Určete lokální minimum funkce:

a) $f(x) = \frac{x^4}{4} - \frac{x^3}{3} - x^2,$ b) $f\ (x) = \frac{x^4+4x^3-8x^2}{4}$

c) $f(x) = \ 10x^3 - \ 6x^5,$ d) $f(x) = x - \frac{1}{2x^2}$

e) $f(x) = \ x^3 - \ 3x \ + \ 2,$ f) $f(x) = \ e^{x^2 + \ 2x \ - \ 4}$

Výsledky:

a) min $\left[-1, -\frac{5}{12}\right], \left[2, -\frac{8}{3}\right]$; b) min $[-4, -32], \left[1, -\frac{3}{4}\right]$; c) min $[-1, -4]$; d) nemá min; e) min [0, 1]; f) $[-1, e^{-5}\ ]$ (Madrová, Mošová 2018)

3.7.3 Příklad

Nalezněte minimum funkce $f(x) = \ (x - 2)^2 + \ 6$

Řešení:

In [27]:

```
grad_sestup_1((x-2)**2 + 6, 1, 0.1)
```

Out [27]:

Iterace 1 Hodnota x: 1.20000000000000

Iterace 2 Hodnota x: 1.36000000000000

Iterace 3 Hodnota x: 1.48800000000000

.

.

.

.

Iterace 53 Hodnota x: 1.99999269249181

Iterace 54 Hodnota x: 1.99999415399345

Iterace 55 Hodnota x: 1.99999532319476

Iterace 56 Hodnota x: 1.99999625855581

Funkce nabývá lokálního minima pro $x = 2$

Kód grafu

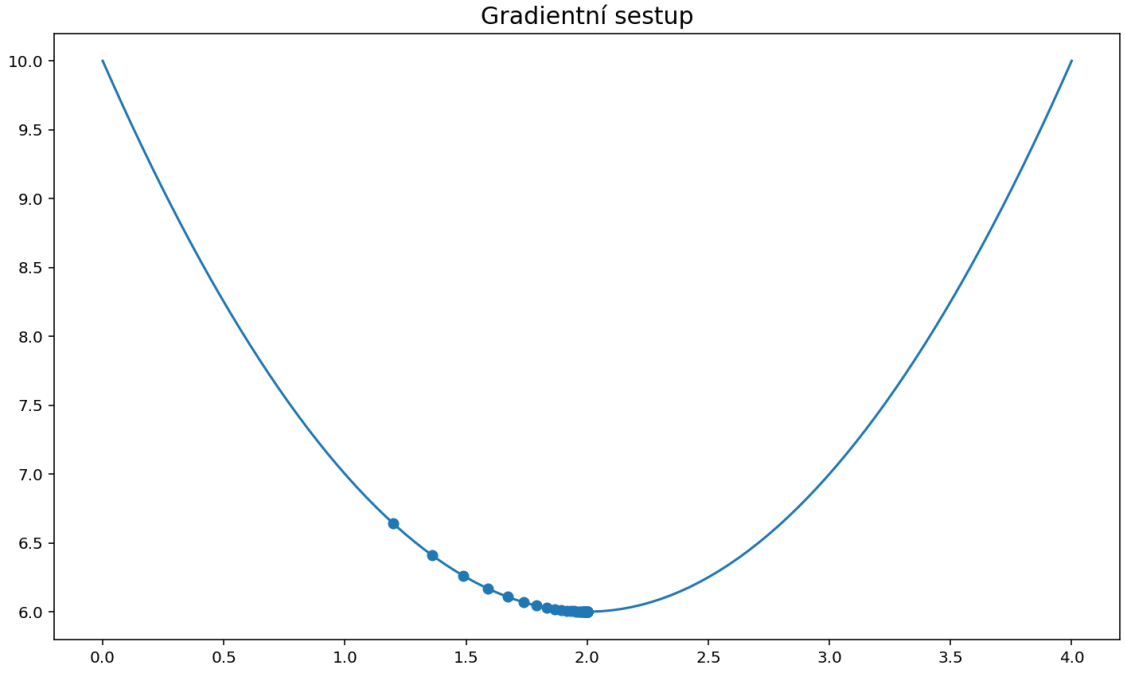
In [28]:

```
f = (x-2)**2 + 6
x = np.linspace(0, 4, 100)
fig, ax = plt.subplots()
plt.title("Gradientní sestup", fontsize=15)
ax.plot(x, f)
ax.scatter(seznam_x,seznam_y)
```

(Vlastní zpracování v softwaru Jupyter Notebook)

Out [28]:

Obrázek č. 11 – Graf funkce $f(x) = (x-2)^2 + 6$ spolu s body reprezentující jednotlivé iterace.



Zdroj: Vlastní zpracování v programu Jupyter Notebook

3.7.4 Kód v Pythonu pro dvě proměnné

Následující kód využívá gradientního sestupu pro nalezení minima funkce o dvou proměnných.

In [29]:

```
def grad_sestup_2(f, x0, y0):
    """
    Funkce grad_sestup_2 představuje metodu gradientního sestupu pro hledání minima funkce
dvou proměnných.

    Příklady
    ==========

    Správně zvolené počáteční body:

    >>>grad_sestup_2((27/10)*x**2*y + (14/10)*y**3 - (69/10)*y - (54/10)*x, 0, 0)
    Iterace 1 Hodnota [x, y]:  [0.540000000000000, 0.690000000000000]
    Iterace 2 Hodnota [x, y]:  [0.878796000000000, 1.10130600000000]
    Iterace 3 Hodnota [x, y]:  [0.896171413908960, 1.07338228903656]
    .
    .
    .
    Iterace 66 Hodnota [x, y]:  [0.999988968602077, 1.00000838478303]
    Iterace 67 Hodnota [x, y]:  [0.999990397824066, 1.00000729845778]
    Iterace 68 Hodnota [x, y]:  [0.999991641869713, 1.00000635288098]
    Funkce nabývá lokálního minima pro [x , y] = [1.0, 1.0]

    Špatně zvolené body:

    >>>grad_sestup_2((27/10)*x**2*y + (14/10)*y**3 - (69/10)*y - (54/10)*x, -1, 3)
    Iterace 1 Hodnota [x, y]:  [1.16000000000000, -0.359999999999999]
    Iterace 2 Hodnota [x, y]:  [1.92550400000000, -0.0877439999999993]
    Iterace 3 Hodnota [x, y]:  [2.55673776840704, -0.402020310589439]
    Iterace 4 Hodnota [x, y]:  [3.65178244475244, -1.54486601367027]
    Iterace 5 Hodnota [x, y]:  [7.23820032238897, -5.45783169017811]
    Iterace 6 Hodnota [x, y]:  [29.1108350360606, -31.4244777835679]
    Iterace 7 Hodnota [x, y]:  [523.638941016031, -674.292548905066]
    Iterace 8 Hodnota [x, y]:  [191190.530512587, -265668.577930705]
    Error. Blížíme se k nekonečnu. Prosím změnte x0 nebo y0

    Zvolení bodů minima:
    >>>grad_sestup_2((27/10)*x**2*y + (14/10)*y**3 - (69/10)*y - (54/10)*x, 1, 1)
    Narazili jste na bod podezřelí z extrému nebo sedlo. Zkuste pozměnit x0 nebo y0 na body v
nejbližším okolí

    Situace viz výše může nastat i když zvolíme maximum nebo sedlo:
    >>>>>>grad_sestup_2((27/10)*x**2*y + (14/10)*y**3 - (69/10)*y - (54/10)*x, -1, -1)
    Narazili jste na bod podezřelí z extrému nebo sedlo. Zkuste pozměnit x0 nebo y0 na body v
```

```
nejbližším okolí

   To lze vyřešit jednoduchým pozměněním x0 nebo y0:
   >>>grad_sestup_2((27/10)*x**2*y + (14/10)*y**3 - (69/10)*y - (54/10)*x, -0.9, -1)
   Iterace 1 Hodnota [x, y]:  [-0.946000000000000, -0.820200000000000]
   Iterace 2 Hodnota [x, y]:  [-0.824990968000000, -0.654373096800000]
   Iterace 3 Hodnota [x, y]:  [-0.576510991063582, -0.327983565988587]
   .
   .
   .
   Iterace 73 Hodnota [x, y]:  [0.999989539951184, 1.00000795051122]
   Iterace 74 Hodnota [x, y]:  [0.999990895146394, 1.00000692045207]
   Iterace 75 Hodnota [x, y]:  [0.999992074757251, 1.00000602385078]
   Funkce nabývá lokálního minima pro [x, y] = [1.0, 1.0]


   Parametry
   ==========

   f:
      - Funkce dvou reálných proměnných
   x0:
      - vstupní hodnota x
      - číslo
      - Počáteční bod, od kterého začínáme sestupovat.
   y0:
      - vstupní hodnota y
      - číslo
      - Počáteční bod y, od kterého začínáme sestupovat.
   """
   xu = x0
   yu = y0
   η = 0.1
   p_f_x = sym.diff(f,x)
   p_f_y = sym.diff(f,y)
   vkx = 1
   vky = 1
   q = 0
   global seznam_x
   global seznam_y
   global seznam_z
   seznam_x = []
   seznam_y = []
   seznam_z = []
   while vkx > 0.000001 and vky > 0.000001:
      xo = x0
      yo = y0
      x0 = x0 - η * p_f_x.subs({x: xo,y:yo})
      y0 = y0 - η * p_f_y.subs({x: xo,y:yo})
      vkx = (abs(x0-xo))
      vky = (abs(y0-yo))
      q = q + 1
```

```
        print("Iterace",q,"Hodnota [x, y]: ",[x0, y0])
        seznam_x.append(x0)
        seznam_y.append(y0)
        if q > 10000:
            break
        elif vkx > 10000 or vky > 10000:
            break
        elif isinstance(x0, sym.core.add.Add) or isinstance(y0, sym.core.add.Add):
            break
        else:
            continue
    seznam_xy = list(zip(seznam_x, seznam_y))
    for bod in seznam_xy:
        z = f.subs([(x,bod[0]),(y,bod[1])])
        seznam_z.append(z)
    if q > 10000:
        display(Math("\\text{V případě, že funkce překročí 10000 iterací, tak se sama zastaví. Pokud se x0 nebo y0 stále zvyšuje nebo snižuje, pak se blížíme k nekonečnu a je třeba zvolit jinné x0 nebo y0.}"))
        display(Math("\\text{Jestli tomu tak není, potom čísla, okolo kterých se hodnoty [x0,y0] pohybují .}"))
    elif q > 2000:
        minx = round(x0, 1)
        miny = round(y0, 1)
        display(Math("\\text{ Funkce nabývá lokálního minima pro } [x, y] = %s" %sym.latex([minx,miny])))
    elif isinstance(x0, sym.core.add.Add):
        display(Math("\\text{Error. Prosím změnte x0 nebo y0}"))
    elif q == 1:
        display(Math("\\text{Narazili jste na bod podezřelí z extrému nebo sedlo. Zkuste pozměnit x0 nebo y0 na body v nejbližším okolí}"))
    elif vkx > 1000 or vky > 1000:
        display(Math("\\text{Error. Blížíme se k nekonečnu. Prosím změnte x0 nebo y0}"))
    else:
        minx = round(x0, 2)
        miny = round(y0, 2)
        display(Math("\\text{ Funkce nabývá lokálního minima pro } [x, y] = %s" %sym.latex([minx,miny])))
```

(Vlastní zpracování v softwaru Jupyter Notebook; Donges, 2021; Crypto1, 2020; StatQuestWithJoshStarmer, 2019)

3.7.5 Testování pro dvě proměnné

Kód byl důkladně otestován na následujících příkladech.

Určete lokální minimum funkce:

a) $f(x,y) = x^2 + 2xy + 3y^2 + 5x + 2y$

b) $f(x,y) = x^3 + xy^2 - 2xy - 5x$

c) $f(x,y) = x^2 + xy + y^2 - 6x - 9y$

d) $f(x,y) = x^2 - xy + y^2 - +9x - 6y + 20$

e) $f(x,y) = e^{2x}(x + y^2 + 2y)$

f) $f(x,y) = (x^2 + y^2)e^{-x^2-y^2}$

Výsledky:

a) min $\left[-\frac{13}{4}, \frac{3}{4}\right]$; b) min $\left[\sqrt{2}, 1\right]$; c) min $[1, 4]$; d) min $[-4, 1]$; e) min $\left[\frac{1}{2}, -1\right]$; f) min $[0, 0]$ (Madrová, 2018; Klaška, 2009)

3.7.6 Příklad

Nalezněte minimum funkce $f(x,y) = xy + \frac{20}{y} + \frac{50}{x}$

Řešení:

In [30]:

```
grad_sestup_2(x * y + 20/y + 50/x, 1, 1)
```

Out [30]:

Iterace 1 Hodnota [x, y]: [5.90000000000000, 2.90000000000000]

Iterace 2 Hodnota [x, y]: [5.75363688595231, 2.54781212841855]

Iterace 3 Hodnota [x, y]: [5.64989328299403, 2.28055092000008]

.

.

.

Iterace 156 Hodnota [x, y]: [5.00008058842092, 1.99998179218629]

Iterace 157 Hodnota [x, y]: [5.00007596228448, 1.99998283737537]

Iterace 158 Hodnota [x, y]: [5.00007160170267, 1.99998382256970]

Funkce nabývá lokálního minima pro $[x, y] = [5.0, 2.0]$

Kód grafu

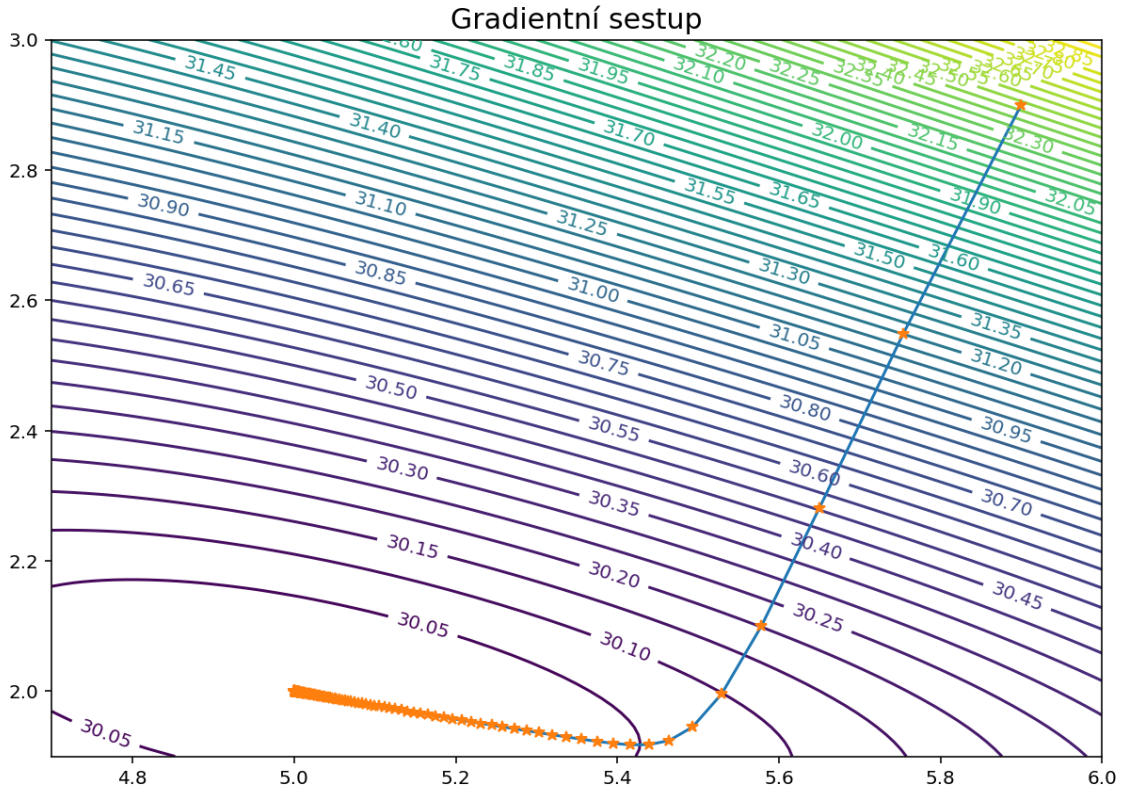
In [31]:

```
def fun(x,y):
    return x * y + 20/y + 50/x
x1 = np.linspace(4.7, 6, 100)
x2 = np.linspace(1.9, 3, 100)
X1, X2 = np.meshgrid(x1, x2)
Z = fun(X1, X2)
fig = plt.figure(figsize = (10,7))
contours = plt.contour(X1, X2, Z, 60)
plt.clabel(contours, inline = True, fontsize = 10)
plt.title("Gradientní sestup", fontsize=15)
plt.plot(np.asarray(seznam_x), np.asarray(seznam_y))
plt.plot(np.asarray(seznam_x), np.asarray(seznam_y), '*')
plt.show()
```

(Vlastní zpracování v softwaru Jupyter Notebook)

Out [31]:

Obrázek č. 12 – Graf funkce $f(x, y) = xy + \frac{20}{y} + \frac{50}{x}$ v konturovém tvaru spolu s body reprezentující jednotlivé iterace a spojnice mezi nimi



Zdroj: Vlastní zpracování v programu Jupyter Notebook

3.8 Newtonova metoda

Newtonova metoda neboli metoda tečen se využívá v numerické matematice k nalezení kořenů funkce nebo k řešení soustavy nelineárních algebraických rovnic.

Newtonovy metody lze využít na derivaci funkce f' na nalezení jejich kořenů ($f'(\mathrm{x}) = 0$), tedy kritických bodů funkce f. (Minima, maxima či sedla.)

Základní tématem optimalizace je minimalizace funkcí.

Nejprve zvažme funkci jedné proměnné.

Mějme funkci f, kterou lze dvakrát zderivovat, a chceme vyřešit optimalizační problém $\min_{x\in\mathbb{R}} f(x)$.

Newtonova metoda řeší tento problém vytvořením posloupnosti $\{x_k\}$ od počátečního bodu $x_0 \in \mathbb{R}$, který konverguje k minimalizátoru x_* funkce f využitím druhého řádu Taylorovy řady, který pro x_k vypadá takto

$$f(x_k + t) \approx f(x_k) + f'(x_k)t + \frac{1}{2}f''(x_k)t^2$$

Člen x_{k+1} definujeme tak, aby minimalizoval výše zmíněnou kvadratickou aproximaci v t, tudíž $x_{k+1} = x_k + t$.

V případě, že druhá derivace je kladná, kravdratická aproximace je konvexní funkcí t, a její minimum získáme položením derivace nule. Pokud platí

$$0 = \frac{d}{dt}\left(f(x_k) + f'(x_k)t + \frac{1}{2}f''(x_k)t^2\right) = f'(x_k) + f''(x_k)t,$$

minima dosáhneme pro

$$t = -\frac{f'(x_k)}{f''(x_k)}.$$

Newtonova metoda pomocí našeho závěru provede iteraci

$$x_{k+1} = x_k + t = x_k - \frac{f'(x_k)}{f''(x_k)}.$$

V případě funkce vyšších řádu lze místo derivace $f'(x_k)$ dosadit gradient $\nabla f(x_k)$ a místo $f''(x_k)$ dosadit Hessovu matici.

V závěru by jednotlivá iterace vypadala takto

$$x_{k+1} = x_k - \left[H_f(x_k)\right]^{-1} \nabla f(x_k).$$

(Wikipedia, 2022; mathematicalmonk, 2011)

3.8.1 Kód v Pythonu pro jednu proměnnou

Následující kód využívá Newtonovy metody pro zjištění minima funkce jedné reálné proměnné

In [32]:

```
def newton_1(f, x0):
    """
    Funkce, která využívá Newtonovy metody pro určení lokálního extrému funkce jedné
proměnné.

    Příklady
    ==========

    Správně zvolené x0:

    >>>newton_1(x**3 - 2*x**2 - 4*x + 5, 1)
    Funkce nabývá lokálního minima pro x = 2.0

    Špatně zvolené x0:

    >>>newton_1(x**3 - 2*x**2 - 4*x + 5, - 1)
    Error. Zvolte jiné x0

    V případě, že funkce nemá extrém tak na výběru x0 nezáleží

    >>>newton_1(ln((x - 2)/(x+1) ), 1.7)
    Error. Zvolte jiné x0

    Vrací i maximum, jestliže zvolíme x0 přímo jako maximum

    >>>newton_1(x**4 - 24*x**2 + 80, 0)
    Funkce nabývá lokálního maxima pro x = 0


    Parametry
    ==========
```

```
    f:
        - Funkce reálné proměnné
    x0:
        - vstupní hodnota x
        - číslo
        - Počáteční bod, od kterého se začínáme pohybovat.
    """
    df = sym.diff(f, x)
    dff = sym.diff(df, x)
    dfff = sym.diff(dff, x)
    xu = x0
    xo = x0 + 0.1
    while abs(x0-xo) > 0.0001:
        xo = x0
        x0 = x0 - (df.subs(x,xo)/dff.subs(x,xo))
        if x0 == zoo or x0 == -zoo:
            break
        elif isnan(x0):
            break
        elif abs(x0) > 100*abs(xu):
            break
        else:
            continue
    minx = round(x0, 2)
    if x0 == zoo or x0 == -zoo:
        display(Math("\\text{Error. Zvolte jiné x0}"))
    elif isnan(x0):
        display(Math("\\text{Error. Zvolte jiné x0}"))
    elif abs(x0) > 100*abs(xu):
        display(Math("\\text{Error. Zvolte jiné x0}"))
    elif dff.subs(x, x0) > 0:
        display(Math("\\text{ Funkce nabývá lokálního minima pro} \; x = %s" %sym.latex(minx)))
    else:
        display(Math("\\text{ Funkce nabývá lokálního maxima pro} \; x = %s" %sym.latex(minx)))
```

(Vlastní zpracování v softwaru Jupyter Notebook;Wikipedia, 2022; mathematicalmonk, 2011)

3.8.2 Testování pro jednu proměnnou

Kód byl důkladně otestován na následujících příkladech.

Určete lokální extrémy funkce:

a) $f(x) = x^4$,

b) $f(x) = x^4 - 24x^2 + 80$

c) $f(x) = \frac{x^2}{x+4}$,

d) $f(x) = \frac{3}{4} ln \frac{x+2}{2-x} - x$

e) $f(x) = \sin(x)\cos(x)$,

f) $f(x) = \arctan(\sqrt{x^2+1})$

Výsledky:

a) min $[0,0]$; b) max $[0,80]$, min $\left[-2\sqrt{3},-64\right]$, $\left[2\sqrt{3},-64\right]$; c) max $[-8,-16]$, min $[0,0]$; d) max $[-1,0.18]$, min $[1,-0.18]$; e) max $\left[\frac{\pi}{4}+k\pi,\frac{1}{2}\right]$, min $\left[-\frac{\pi}{4}+k\pi,-\frac{1}{2}\right]$; f) min $\left[0,\ \frac{\pi}{4}\right]$ (Hamříková, 2007; Kuben, Šarmonová, 2006)

3.8.3 Příklad

Nalezněte minimum funkce $f(x) = x^2 - 3x + 4$

In [33]:

```
newton_1(x**2 - 3*x + 4, 1)
```

Out [33]:

Funkce nabývá lokálního minima pro $x = 1.5$

Kód grafu

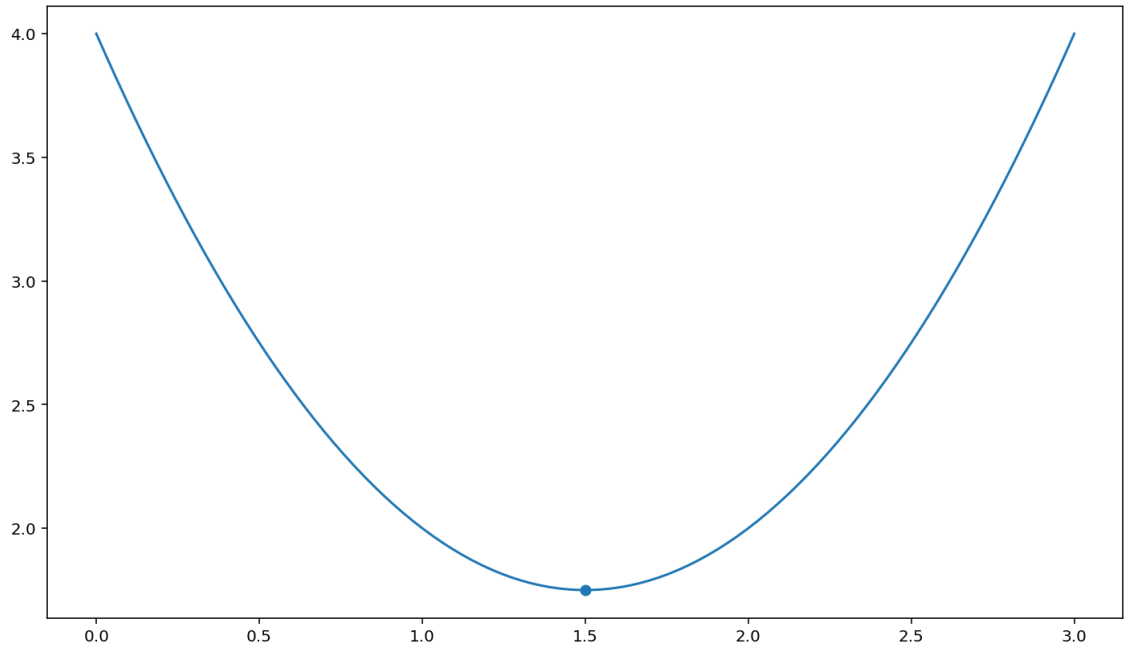
In [34]:

```
f = x**2 - 3*x + 4
x = np.linspace(0, 3, 100)
fig, ax = plt.subplots()
ax.plot(x, f)
ax.scatter(1.5,1.75)
```

(Vlastní zpracování v programu Jupyter Notebook)

Out [34]:

Obrázek č. 13 – Graf funkce $f(x) = x^2 - 3x + 4$ a bodu reprezentující její minimum



Zdroj: Vlastní výroba v programu Jupyter Notebook

3.8.4 Kód v Pythonu pro dvě proměnné

Následující kód pomocí Newtonovy metody spočítá minimum funke dvou reálných proměnných.

In [35]:

```
def newton_2(f, x0, y0):
    """
    Funkce, která využívá Newtonovy metody pro určení lokálního extrému funkce dvou proměnných.

    Příklady
    ==========

    Správné zadání počátečních bodů:

    >>>newton_2(4*x - y + (1/x) - (1/y), 1,1)
    Funkce nabývá lokálního maxima pro [x,y] = [−0.5, 1]

    >>>newton_2(4*x - y + (1/x) - (1/y), 0.6,-0.8)
    Funkce nabývá lokálního minima pro [x,y] = [0.5, −1.0]

    Špatně:

    >>>newton_2(4*x - y + (1/x) - (1/y), 0,1)
    Funkce nemá extrém v okolí [x,y], Změnte [x0,y0]

    Někdy se vám může stát, že jste blíže výsledku než si myslíte:

    >>>newton_2(ln(x**2 + y**2 + 2*x - 4*y + 14), -1, 1.5)
    Funkce nemá extrém pro [x,y] = [−1, 2.03]

    Stačí pozměnit jeden z počátečních bodů

    >>>newton_2(ln(x**2 + y**2 + 2*x - 4*y + 14), -1, 1.8)
    Funkce nabývá lokálního minima pro [x,y] = [-1, 2.0]

    Mužeme narazit i na stacionární body kde funkce nemá extrém:

    >>>newton_2(4 - x**3 - y*x**2 - 3*y**2 + 20*y , 1,2)
    Funkce nemá extrém pro [x,y]=[0, 3.33]

    Parametry
    ==========
    f:
       - Funkce, jejíž minimum chceme znát.
    x0:
       - vstupní hodnota x
       - číslo
```

```
    - Počáteční bod, od kterého se začínáme pohybovat.
  y0:
    - vstupní hodnota y
    - číslo
    - Počáteční bod, od kterého se začínáme pohybovat.
  """
  gradf = derive_by_array(f,[x,y])
  Hmat = derive_by_array(gradf, [x,y])
  m = Matrix(Hmat)
  HmatInv = m.inv(method="LU")
  n = Matrix(gradf)
  odhL = [x0,y0]
  odhMat = Matrix(odhL)
  vkx = 1
  vky = 1
  while vkx > 0.000001 and vky > 0.000001:
    xyo = odhMat
    odhMat = odhMat - (HmatInv.subs([(x,xyo[0]),(y,xyo[1])])*n.subs([(x,xyo[0]),(y,xyo[1])]))
    if isnan(odhMat[0]) or isnan(odhMat[1]):
      break
    vkx = (abs(odhMat[0]-xyo[0]))
    vky = (abs(odhMat[1]-xyo[1]))
  if isnan(odhMat[0]) or isnan(odhMat[1]):
    display(Math("\\text{Nemá extrém v okolí [x,y], Změnte [x0,y0]}"))
  else:
    xmin = round(odhMat[0], 2)
    ymin = round(odhMat[1], 2)
    ms = Hmat.subs([(x, xmin), (y, ymin)])
    sm = Matrix(ms)
    ss = gradf.subs([(x, xmin), (y, ymin)])
    if ms[0][0] == 0:
      display(Math("\\text{Funkce nemá extrém pro } [x,y] = %s" %sym.latex([xmin,ymin])))
    elif sm.det() <= 0 :
      display(Math("\\text{Funkce nemá extrém pro  } [x,y] = %s" %sym.latex([xmin,ymin])))
    elif ss[0] != 0 or ss[1] != 0:
      display(Math("\\text{Funkce nemá extrém pro  } [x,y] = %s" %sym.latex([xmin,ymin])))
    elif ms[0][0] < 0:
      display(Math("\\text{Funkce nabývá lokálního maxima pro } [x,y] = %s"
%sym.latex([xmin,ymin])))
    else:
      display(Math("\\text{ Funkce nabývá lokálního minima pro  } [x,y] = %s"
%sym.latex([xmin,ymin])))
```

(Vlastní zpracování v programu Jupyter Notebook; Wikipedia, 2022; mathematicalmonk, 2011)

3.8.5 Testování pro dvě proměnné

Kód byl důkladně otestován na následujících příkladech.

Určete lokální extrémy funkce:

a) $f(x,y) = 4x^2 + 5y^2 - 12x + 15y + 6$

b) $f(x,y) = 3x^2 - 4xy + 6x + 4y^2 - 4y + 9$

c) $f(x,y) = e^{x+y}(x^2 + y^2)$

d) $f(x,y) = 3y^3 - 6xy + y^2 + 3$

e) $f(x,y) = x^3 - 4xy + y^2 + 4x + 1$

f) $f(x,y) = x\sqrt{y} - x^2 - y + 6x + 5$

Výsledky:

a) min $\left[\frac{3}{2}, -\frac{3}{2}, \frac{57}{4}\right]$; b) min $[-1, 0, 6]$; c) min $[0, 0, 0]$; d) nemá extrém; e) min $[2, 4, 1]$; f) max $[4, 4, 17]$ (Hamříková, 2007)

3.8.6 Příklad

Nalezněte minimum funkce $f(x,y) = x - y + 2x^2 + 2xy + y^2$

In [36]:

```
newton_2(x - y + 2*x**2 + 2*x*y + y**2, 0, 0)
```

Out [36]:

Funkce nabývá lokálního minima pro $[x,y] = [-1, 1.5]$

Kód grafu

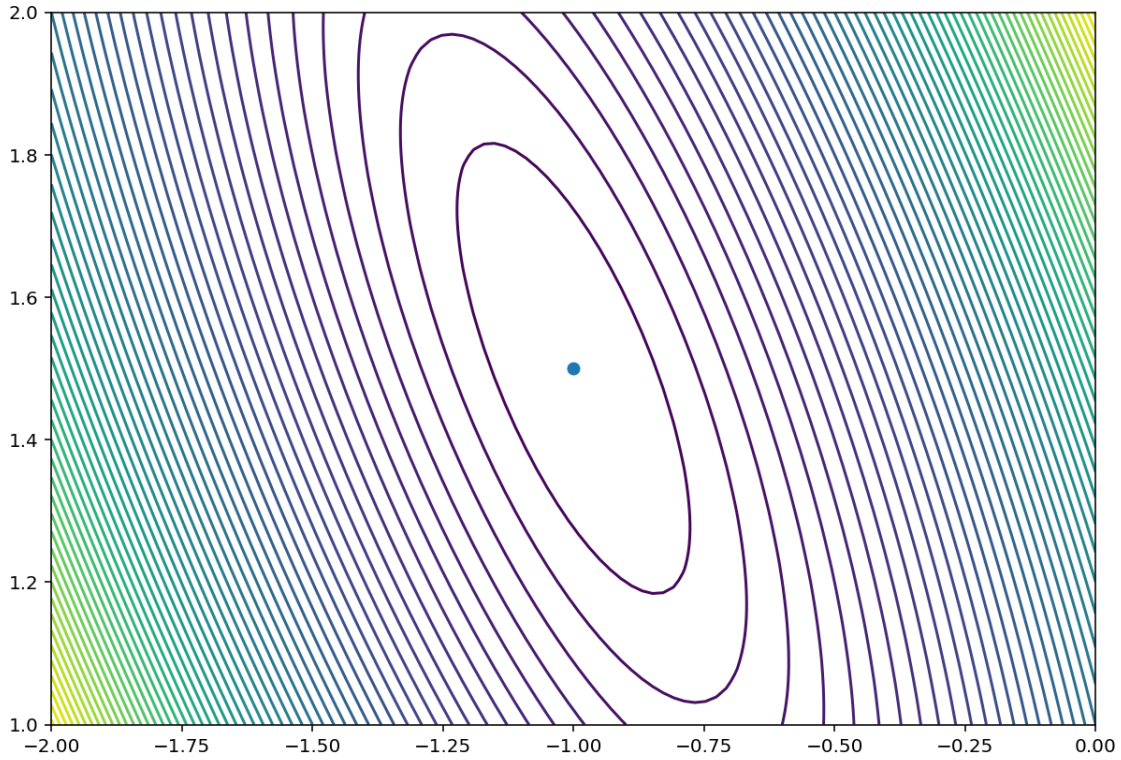
In [37]:

```
def fun(x,y):
    return x - y + 2*x**2 + 2*x*y + y**2
x1 = np.linspace(-2, 0, 100)
x2 = np.linspace(1, 2, 100)
X1, X2 = np.meshgrid(x1, x2)
Z = fun(X1, X2)
fig = plt.figure(figsize = (10,7))
contours = plt.contour(X1, X2, Z, 60)
plt.scatter(-1,1.5)
```

(Vlastní zpracování v programu Jupyter Notebook)

Out [37]:

Obrázek č. 14 – Graf funkce $f(x, y) = x - y + 2x^2 + 2xy + y^2$ v konturovém tvaru spolu s bodem, který představuje minimum f(x,y)



Zdroj: Vlastní spracování v programu Jupyter Notebook

Závěr

V soudobém věku informačních technologií klademe především důraz na automatizaci. Vzhledem k tomu, že matematika se vyskytuje prakticky všude okolo nás, je velmi důležité provádět správné výpočty svižně, zejména když řešíme složitější problémy, než je jednoduchá násobilka.

Vytvářet algoritmy postupných řešení matematických problémů dnes může skoro každý z důvodu volné dostupnosti interpretů programovacích jazyků, které jsou ke stažení zdarma na internetu.

Cílem mé práce bylo prezentovat některé možnosti knihovny Sympy při řešení úloh z teorie matematické optimalizace. Tuto knihovnu lze využít skrze syntax programovacího jazyku Python. Měl jsem navíc úkazat prezentační možnosti knihoven Pythonu pro zobrazení dat a matematických formulí.

Napsal jsem algoritmy pro hledání lokálních a globálních extrémů funkce jedné reálné proměnné, lokálních, vázaných a globálních funkcí dvou reálných promenných, algoritmy, které využívají metody gradientního sestupu pro hledání minim funkcí jedné a dvou reálných proměných, a algoritmy Newtonovy metody v rámci teorie optimalizace.

Prezentační možnosti Sympy jsem předvedl ve formě grafického zobrazení zadaných funkcí a jejich řešení. Nadále jsem prezentoval tyto možnosti pomocí dalších dvou knihoven NumPy a Matplotlib. Jeden z grafů jsem vytvořil pomocí softwaru Sagemath.

Myslím si, že svůj cíl jsem tedy splnil, a doufám, že funkce, které jsem zpracoval, budou přinosem při řešení úloh z teorie optimalizace.

Seznam použitých zdrojů

Literatura:

1. BACHMAN, David. Advanced calculus demystified. New York: McGraw-Hill, c2007. ISBN 978-0-07-148121-2.
2. DOŠLÁ, Zuzana a Jaromír KUBEN. Diferenciální počet funkcí jedné proměnné. Brno: Masarykova univerzita, 2003. ISBN isbn80-210-3121-2.
3. HAMHALTER, Jan a Jaroslav TIŠER. Diferenciální počet funkcí více proměnných. Vyd. 2. Praha: Česká technika - nakladatelství ČVUT, 2005, c1997. ISBN 80-01-03356-2.
4. HAMŘÍKOVÁ, Radka. Sbírka úloh z matematiky. Ostrava: VŠB - Technická univerzita Ostrava, 2007. ISBN 978-80-248-1317-2.
5. KALUS, René a Daniel HRIVŇÁK. Breviář vyšší matematiky. Ostrava: Ostravská univerzita, 2001. ISBN 80-7042-819-8.
6. KLAŠKA, Jiří. Diferenciální a integrální počet funkcí více proměnných. Sbírka řešených příkladů. Brno: Fakulta strojního inženýrství, 2009.
7. KREML, Pavel. Matematika II. Ostrava: VŠB - Technická univerzita Ostrava, 2007. ISBN 978-80-248-1316-5.
8. KUBEN, Jaromír a Petra ŠARMANOVÁ. Diferenciální počet funkcí jedné proměnné. Ostrava: VŠB - Technická univerzita, 2006. ISBN 80-248-1192-8.
9. KUHLMAN, Dave. A Python Book. Beginning Python, Advanced Python, and Python Exercises. SCL, 2009.
10. MADROVÁ, Vladimíra a Vratislava MOŠOVÁ. Diferenciální počet (1). Funkce jedné proměnné. Řešené příklady.Olomouc: Moravská vysoká škola Olomouc, 2018. ISBN 978-80-7455-087-4.
11. MORÁVKOVÁ, Zuzana. Matematika II: pracovní listy. Ostrava: Vysoká škola báňská - Technická univerzita Ostrava, 2014. ISBN 978-80-248-3324-8.

12. PILGRIM, Mark. Ponořme se do Python(u) 3: Dive into Python 3. Praha: CZ.NIC, c2010. CZ.NIC. ISBN 978-80-904248-2-1.

Webové stránky:

1. (ML 15.2) Newton's method (for optimization) in multiple dimensions [online]. 2011 [cit. 2021-11-18]. Dostupné z WWW: https://www.youtube.com/watch?v=42zJ5xrdOqo. Kanál uživatele mathematicalmonk.

2. 27 - Vázané extrémy a Lagrangeova funkce (MAT - Diferenciální počet funkcí více proměnných) [online]. 2017 [cit. cit. 2021-07-24]. Dostupné z WWW: https://www.youtube.com/watch?v=atLNVLdvNkA. Kanál uživatele Isibalo.

3. 28 - Globální (absolutní) extrémy (MAT - Diferenciální počet funkcí více proměnných) [online]. 2017 [cit. 2022-01-21]. Dostupné z WWW: https://www.youtube.com/watch?v=Yzw-LRu3oBs. Kanál uživatele Isibalo.

4. Finding Local Maximum and Minimum Values of a Function - Relative Extrema [online]. 2018 [cit. 2021-07-19]. Dostupné z WWW: https://www.youtube.com/watch?v=WCq3sRzsJfs. Kanál uživatele The Organic Chemistry Tutor.

5. Globální extrémy | 11/13 Funkce více proměnných | Matematika | Onlineschool.cz [online]. 2017 [cit. 2022-01-21]. Dostupné z WWW: https://www.youtube.com/watch?v=Fivz60pN6WQ. Kanál uživatele Onlineschool cz

6. Gradient Descent, Step-by-Step [online]. 2019 [cit. 2021-11-18]. Dostupné z WWW: https://www.youtube.com/watch?v=sDv4f4s2SB8. Kanál uživatele StatQuest with Josh Starmer

7. Gradient Descent: An Introduction to 1 of Machine Learning's Most Popular Algorithms – Niklas Donges [online]. 2021 [2021-08-27]. Dostupné z WWW: https://builtin.com/data-science/gradient-descent

8. How Does the Gradient Descent Algorithm Work in Machine Learning? – Crypto1 [online]. 2020 [cit. 2021-11-18]. Dostupný z WWW: https://www.analyticsvidhya.com/blog/2020/10/how-does-the-gradient-descent-algorithm-work-in-machine-learning/

9. Import module in Python [online] - GeekforGeeks. 2021 [cit. 2022-02-5]. Dostupný z WWW: https://www.geeksforgeeks.org/import-module-python/

10. Matplotlib 3.5.0 documentation [online]. 2021 [cit. 2022-02-10]. Dostupné z WWW: https://matplotlib.org/3.5.0/

11. Maxima, minima, and saddle points (article) - Khan Academy[online]. 2021 [cit. 2021-12-13]. Dostupné z WWW: https://www.khanacademy.org/math/multivariable-calculus/applications-of-multivariable-derivatives/optimizing-multivariable-functions/a/maximums-minimums-and-saddle-points

12. Newton's method in optimization [online]. 2021 [cit. 2021-11-18]. Dostupné z WWW: https://en.wikipedia.org/wiki/Newton%27s_method_in_optimization

13. NumPy Documentation [online]. 2022 [cit. 2022-02-10]. Dostupné z WWW: https://numpy.org/doc/

14. Obsah trojúhelníku [online]. 2022 [2022-01-22]. Dostupné z WWW: https://www.matweb.cz/obsah-trojuhelniku/

15. Python 3.10.4 documentation [online]. 2022 [cit. 2022-02-10]. Dostupný z WWW: https://docs.python.org/3/

16. SymPy [online]. 2021 [cit. 2022-02-10]. Dostupný z WWW: https://www.sympy.org/en/index.html

17. SymPy Documentation [online]. 2022 [cit. 2022-02-10]. Dostupné z WWW: https://docs.sympy.org/latest/index.html

18. The Making of Python: A Conversation with Guido van Rossum, Part I - Venners [online]. 2003 [cit. 2022-02-10]. Dostupný z WWW: https://www.artima.com/articles/the-making-of-python