Teoretická a počítačová chemie
cvičení

Libor Veis
Andrej Antalík

Obsah

1 Úvod do Unixu

1.1 Nápověda

`man příkaz` nápověda k danému příkazu

implicitní prohlížeč je program `less`:

pohyb v prohlížeči	**PgUp/PgDown**
hledání	**/ text**
ukončení	**q**

1.2 Práce se soubory a adresáři

`pwd`	vypíše cestu k aktuální pozici
`ls`	zobrazí obsah daného adresáře
`ls -l`	podrobnější výpis
`ls -l soubor`	informace o daném souboru
`ls -lrt`	řazení dle data poslední změny, nejnovější na konci
`ls -la`	zobrazení obsahu včetně skrytých souborů (začínají tečkou)
`diff soubor1 soubor2`	*textové* porovnání dvou souborů (po řádcích)
`mv`	přejmenuje nebo přesune soubory/adresáře
`rm -rf`	rekurzivní mazání (včetně obsahu podadresářů)
`cp`	kopírování
`cp -r`	kopírování adresářů včetně obsahu (rekurzivně)
`cat soubor [soubor2...]`	vypíše obsah souboru nebo spojí a vypíše více souborů
`cat soubor | less`	prohlížení souborů
`paste soubor1 soubor2`	sloupcové spojení souborů
`nl soubor`	očísluje řádky souboru a vypíše je
`mkdir`	vytvoří adresář zadaného jména
`cd adresář`	vstup do daného adresáře
`cd ..`	o úroveň výš
`cd`	návrat do domovského adresáře (/home/student)
`chmod a+x`	přidá *všem* uživatelům právo spouštět soubor

1.3 Práce s textem

`sort`	řazení řádků dle abecedy
`head -n N`	vypíše prvních N řádků souboru
`tail -n N`	vypíše posledních N řádků souboru
`tail -f`	vypisuje průběžně přibývající řádky souboru
`grep "řetězec"soubor`	vypíše řádku souboru obsahující řetězec
`grep "řetězec"*`	vypíše jména souborů (z daného adresáře) a řádky obsahující řetězec

1.4 Příkazy pro práci na vzdáleném počítači

`ssh uživatel@počítač`	přihlášení uživatele na vzdálený počítač
`ssh uživatel@počítač -X`	při použití grafického rozhraní
`scp soubor uživatel@počítač:cesta`	zkopíruje soubor na vzdálený počítač, na danou cestu
`scp uživatel@počítač:cesta/soubor .`	zkopíruje vzdálený soubor do aktuálního adresáře (tečka)
`scp uživatel1@počítač1:cesta_k_souboru1 uživatel2@počítač2:cesta2`	obecné použití: ze vzdáleného počítače na jiný

1.5 Příkazy bashe

1.5.1 Přesměrování, roury

`program > soubor`	přesměruje výstup programu do souboru (založí nový)
`program >> soubor`	přesměruje výstup programu na konec souboru (zachová původní obsah)
`program 2> soubor`	přesměruje chybový výstup programu do souboru
`program < soubor`	přesměruje soubor na standardní vstup programu
`program1 | program2`	**roura** - předá výstup jednoho programu na vstup druhého

1.5.2 Řídící struktury

cyklus for přes výčet

```
for promenna in [seznam]; do
  příkazy
done
```

cyklus přes všechny soubory v adresáři

```
for soubor in *; do
  echo $soubor
done
```

cyklus while

```
while [ podmínka ]; do
  příkazy
done
```

podmíněný příkaz	`if [ test ]; then`
	`  příkazy`
	`fi`

proměnné	$název_proměnné
parametry skriptu	$1, $2, ...
počet parametrů	$#
jméno skriptu	$0
první řádka skriptu (interpret shellu)	`#!/bin/bash`
spouštění skriptu	`./skript`

1.5.3 Testování hodnot

testování souborů

`[ -e název ]`	zda existuje soubor nebo adresář daného jména
`[ -d název ]`	zda existuje adresář daného jména
`[ -f název ]`	zda existuje soubor daného jména

testování logických hodnot

`[ (výraz) ]`	výraz je true
`[ !výraz ]`	výraz je false
`[ výraz1 -a výraz2 ]`	logické AND
`[ výraz1 -o výraz2 ]`	logické OR

testování řetězců

`[ string1 = string2 ]`	řetězce jsou ekvivalentní
`[ string1 != string2 ]`	řetězce nejsou ekvivalentní

testování číselných hodnot

`[ číslo1 -eq číslo2 ]`	čísla jsou shodná
`[ číslo1 -ne číslo2 ]`	čísla nejsou shodná
`[ číslo1 -gt číslo2 ]`	větší než
`[ číslo1 -ge číslo2 ]`	větší nebo rovno
`[ číslo1 -lt číslo2 ]`	menší než
`[ číslo1 -le číslo2 ]`	menší nebo rovno

1.6 Klávesové kombinace

Ctrl+c	ukončení programu
Ctrl+d	ukončení standardního vstupu, vypnutí terminálu, odhlášení od vzdáleného počítače
↑, ↓	pohyb v historii příkazů
Tab	doplní příkaz/cestu

1.7 Další užitečné příkazy

`top`	seznam procesů podle vytížení procesoru a paměti
`kill`	zabíjení procesů
`date`	datum a čas
`cal`	kalendář
`clear`	mazání obrazovky

1.8 Editor VIM

i	vstup do režimu psaní
Esc	režim prohlížení
:w!	vynucené uložení
:q!	vynucené vypnutí
yy	kopírování řádku
p	vkládání
dd	mazání řádku
x	mazání znaku
:N	přesun na řádku N
/ text	hledání

1.9 Úlohy

1. Vyzkoušejte si ovládání editoru VIM.
2. Vyzkoušejte si práci se soubory a adresáři: tvorba, kopírování, mazání, přejmenovávání.
3. V případě adresářů zjistěte rozdíl mezi: `ls -l adresář` a `ls -ld adresář`.
4. Vyzkoušejte si přihlášení na vzdálený server (přihlašovací údaje dostanete).
5. Vyzkoušejte si kopírování ze vzdáleného serveru (budete instruováni).
6. Dostanete soubor *h2.log*, jedná se o výpočet molekuly vodíku metodou HFSCF (budeme se učit později), energie molekuly je na stejné řádce jako řetězec E(RHF), zjistěte ji, zjistěte číslo této řádky.
7. Napište skript, který všem souborům ve vašem domovském adresáři přidá koncovku ".backup".
8. Napište skript, který vám řekne, jestli daný soubor (zadaný jako parametr) v daném adresáři (zadaný jako druhý parametr) existuje a připadně, jakou má velikost.
9. V nápovědě příkazu `wc` zjistěte, co dělá přepínač `-l`, použijte ho s pomocí příkazu `ls -l` a roury a zjistěte, kolik je souborů v adresáři */opt/g03.c02/g03*.
10. Řekněte si o input file k programu *Aces II* a napište skript pro výpočet disociační křivky dvouatomové molekuly (např. molekuly vodíku).

11. Zkuste odhadnout, co dělá následující sekvence příkazů a jaký bude obsah jednotlivých proměnných (pokročilejší): `ls=ls; lsls=${#ls}; ls -ls <(ls) >$ls; ls="ls -ls ‘ls‘"; ((lsls--)); [[ $lsls < ${#ls}/${#ls} ]] && ${ls/‘ls‘/ls*} || ${ls/-ls//}`

2 Minimum jazyka Python

Python je objektově orientovaný interpretovaný programovací jazyk, který se v dnešní době využíva v mnoha různých oblastech. Díky absenci nutnosti kompilace a jeho elegantní syntaxi je vhodný jak pro přípravu skriptů automatizujících jednoduché úkoly, tak i pro psaní komplexních programů. V následující části uvádíme stručný přehled základních vlastností tohoto jazyka. Pro podrobnejší přehled doporučujeme podívat se do tutoriálu [2].

2.1 Struktura kódu

Zde uvádíme ukázku, některé klíčové vlastnosti a konvence kódu psaného v Pythonu.

- Skripty mají příponu .py
- Je důležité dbát na verzi interpreteru, protože mezi nimy můžou být výrazné rozdíly v syntaxi. My budeme používat verzi 3.x.
- Jednotlivé bloky kódu (v jiných jazycích označené např. složenými závorkami) se značí odsazením, standardně jednu úroveň tvoří 4 mezery.
- Komentáře v kódu jsou značeny mřížkou # – zbytek řádku za mřížkou se nevykonáva.

```
#!/usr/local/intel/intelpython3/bin/python3.6   # volba interpreteru (příkaz pro shell)
import math                                     # import modulu
import sys

def hello_world():                              # definice funkce
    print('Hello world!')

hello_world()                                   # volání funkce
```

Output:

```
Hello world!
```

2.2 Proměnné

Narozdíl od jiných jazyků není třeba deklarovat typ proměnné, ale požívaná konvence je mít jeden název proměnné pouze na jeden typ.

```
a = 10                          # přiřazení celočíselných proměnných
b = 13
```

```
a = a + b
predmet = 'Teoreticka chemie'   # přiřazení textových řetězců
typ = 'cviceni'
print(a)                        # výpis proměnné na stadardní výstup
print(predmet + ' - ' + typ)    # vypíše: 'Teoreticka chemie - cviceni'
```

Při přiřazení hodnoty se nejprve vykoná operace vpravo od operátoru přiřazení =, a až následně se hodnota uloží do proměnné.
Reálná čísla je možné zadávat několika různými způsoby:

```
0.0035
3.5e-3
```

2.3 Aritmetické operace

Operace +, - a * mají svůj klasický význam, avšak rozlišujeme několik druhů operace dělení. Operátor / značí obecné dělení a v případě neceločíselného podílu vrací reálné číslo. Operátor // vrací celočíselný násobek dělitele v dělenci a operátor modulo % pak zbytek po dělení.

```
a = 5 / 2   # vrací 2.5
b = 5 // 2  # vrací 2
c = 5 % 2   # vrací 1
d = x**2    # operátor mocniny: vrací kvadrát x
```

Stejně jako v jazyce C můžeme použít operace +=, -=, *= a /=, které mají význam simultánní operace a přiřazení. V následujícím kódu jsou uvedeny příklady jejich použiti a v komentáři jejich význam.

```
i += 5    # i = i + 5
i -= 4    # i = i - 4
i *= 2    # i = i * 2
```

Narozdíl od jazyka C však neexistuje operace pro inkrement a je teda nutno použít konstrukce i += 1.

2.4 Funkce

Příklad funkce byl uveden již v úvodní ukázce. Tato funkce nebrala žádné argumenty a nevracela žádnou hodnotu, pouze vykonala v ní zadané operace. V případě, že chceme funkci s argumenty:

```
def scitani(a=4, b=5):
    return a + b

scitani()       # vrací 9
scitani(10)     # vrací 15
scitani(b=10)   # vrací 14
scitani(1, 1)   # vrací 2
```

Python obsahuje také možnost definovat si jednoduché, tzv. *lambda funkce* ve tvaru

```
f = lambda x : x**2
print(f(10)) # vypíše 100
```

2.5 Datové struktury

Zde uvádíme dvě základní datové struktury a to tuple (neboli ntici) a list. Další praktické struktury v Pythonu jsou množiny (set) a slovníky (dictionary). V případě zájmu opět odkazujeme na tutoriál [2]

List

Struktura list plní funkci pole, ve kterém jsou uloženy hodnoty. List je vždy indexovaný od nuly.

```
l = [4, 9, 3, 6, 8]

l[3]                    # 6
l[2:]                   # [3, 6, 8]
l[1:3]                  # [9, 3, 6]
l.append(9)             # [4, 9, 3, 6, 8, 9]
del l[3]                # [4, 9, 3, 8, 9]
len(l)                  # délka listu – 5

matrix = [[1, 2, 3],  # matice zkonstruována formou dvourozměrného list (list listů)
          [4, 5, 6],
          [7, 8, 9]]
matrix[2][1]            # 8

sorted(l)               # vrací seřazený list tj. [3, 4, 8, 9, 9] (l zůstává nezměnen)
l.sort()                # seřadí list l (seřazený list uložený v l)
```

Tuple

Tuple, neboli n-tice je podobně jako list pole hodnot, které je indexováno od nuly. Na rozdíl od listu však není možné po přiřazení měnit hodnotu konkrétního prvku pomocí operátoru přiřazení. Zde uvádíme jen základní použití, avšak upozorňujeme, že možnosti použití této struktury jsou daleko širší.

```
tup = (12, 34, 65, 'test')
tup[2]                          # vrací 65
a, b, _, txt = tup              # a=12, b=34, 65 nejde nikam a txt='test'
```

V uvedeném příkladě značí podtržítko hodnotu, která je při čtení tuple zahozena.
Dále, pokud máme definovanou funkci, která vrací několik hodnot, tak tímto výstupem je právě tuple.

```
def mocniny(x):
    return x**2, x**3
ctverec, krychle = mocniny(10)   # ctverec=100, krychle=1000
```

Naopak, pokud chceme použít tuple jako několik argumentů pro jednu funkci, rozbalíme je pomocí hvězdičky.

```
def soucet(a, b):
    return a+b
tup = (4, 5)
soucet(*tup)        # vrací 9
```

2.6 Podmínky

Podmíněné příkazy mají následující tvar, kde se za if očekává hodnota booleovské preměnné tj. True nebo False.

```
podminka = 5 < 3
slozena = 3 != 10 and 4 > 5

if podminka:
    prikaz
elif slozena:
    prikaz
else:
    prikaz
```

Nejběžnějším výrazem je srovnání čísel, pro které slouží výrazy <, <=, >=, >, rovná se == a nerovná se !=. Dále možno jednotlivé výrazy skládat do složených výroků pomocí logických operátorů **and** a **or**.
V případě množin a listů můžeme testovat, jestli nějaký prvek do nich patří.

```
mnozina = {1, 5, 6, 10}
if 6 in mnozina:
    prikaz
```

Dalším užitečným výrazem je tzv. *ternary operator*, pomocí kterého můžeme přiřadit hodnotu proměnné na základě jednořádkového podmíněného příkazu.

```
a = 5 if x>2 else 10
```

V uvedeném případě se do proměnné *a* uloží 5 když je *x* větší jako 2, jinak 10.

2.7 Cykly

Cykly se používají pro opakující se operace. V případě, že předem známe počet kroků, které je potřeba vykonat, použijeme *for cyklus*:

```
for i in range(5):
    print(i)          # vypíše čísla 0, 1, 2, 3, 4

range(do)             # 0, 1, ..., do-1
range(od, do)         # od, ..., do-1
range(od, do, ink)    # od, od+ink, od+2*ink, ..., do-ink
range(0, 5, 2)        # 0, 2, 4
```

Dále je možné pomocí for cyklu procházet prvky iterovatelných struktur, jakou je například list,

```
l = ['jeden', 'dva', 'tři']
for i in l:
    print(i)              # vypíše postupně jeden, dva a tři
```

případně lze jednoduše vyplnit list pomocí tzv. *list comprehensions*:

```
l = [i**2 for i in range(6)]            # l = [0, 1, 4, 9, 16, 25]
l = [i if i%4 == 0 for i in range(10)]  # l = [0, 4, 8]
```

V druhém uvedeném příkladu jsou z čísel 0–10 v listu vypsány pouze ty, které jsou dělitelné 4 (viz ternary operator).
Když počet kroků neznáme, používáme *while cyklus*, ve kterém je příkaz vykonáván, dokud je podmínka splněna:

```
while podminka:
    prikaz
```

2.8 Moduly a balíky

V případě, že chceme použít funkce nebo třídy z jiných souborů, je třeba je importovat. Takovýmto souborům se říká moduly a může se jednat o náš vlastní kód v souboru s koncovkou *.py*, standardní modul, který je součastí instalace Pythonu nebo modul, který je doinstalován jako součást nějakého balíku.
Ukážeme si možné způsoby importu na příkladu třídy *array* z balíku pro práci s maticemi *NumPy* (viz níže).

```
import numpy
a = numpy.array([1, 2, 3])
```

Po importu k funkcím a třídam přistupujeme jako **modul.funkce()**. V případě, že je jméno příliš dlouhé a v kódu jej používáme často, můžeme si jej přejmenovat.

```
import numpy as np
a = np.array([1, 2, 3])
```

Také je možné importovat jen specifické funkce, nebo všechny funkce (*) z daného modulu.

```
from numpy import array
from numpy import *
a = array([1, 2, 3])
```

Zejména poslední způsob se však nedoporučuje, protože jména importovaných funkcí můžou být v konfliktu s jmény funkcí, které jsou použité v kódu. Doporučujeme proto používat jednu z prvních dvou možností, které umožní předejít případným nejednoznačnostem.

2.9 Čtení a zápis do souboru

Soubor otevřeme pomocí příkazu *open*, kde první argument je jméno souboru a druhý specifikuje, v jakém módu ho chceme otevřít. Základní možnosti jsou: *'r'* pro čtení, *'w'* pro zápis (přepíše soubor) a *'a'* pro zápis na konec souboru (append). Standardně se předpokládá, že se jedná o textový soubor. V případě, že chceme pracovat se souborem binárním, přidáme k uvedeným módům *'b'* tedy např. *'rb'*.
Funkce pro čtení souboru:

```
f = open('jmeno_souboru', 'r')    # otevře textový soubor pro čtení
f.readline()                      # přečte řádek souboru a vrátí jeho obsah
f.readlines()                     # přečte soubor a vrátí list jeho řádků
f.read()                          # přečte soubor a vrátí ho jako jeden řetězec
f.close()                         # zavře soubor
```

Funkce pro zápis do souboru:

```
f = open('jmeno_souboru', 'w')   # otevře textový soubor pro zápis
f.write('test zapisu\n')         # zapíše řetězec 'test zapisu' a začne nový řádek
f.close()
```

Otevřít a procházet soubor lze také pomoci konstrukce *with*, kde soubor zůstáva otevřen pro příkazy vnořené do této konstrukce. V momentě vynoření se soubor zavře stejně, jako kdybychom použili příkaz close(). Procházení souboru po řádkách pak můžeme zapsat nasledovně:

```
with open('jmeno_souboru', 'r') as f:  # otevře soubor pro čtení
    for line in f:                     # prochází soubor po řádcích a vypisuje je
        print(line)
print('tady je uz soubor zavřený')
```

2.10 Další

Konverze typů

Pro změnu datového typu nebo struktury lze jednoduše na proměnnou zavolat konverzní funkci, která je pojmenována podle výsledného typu.

```
a = '50'            # string
int(a)              # celé číslo 50
float(a)            # reálné číslo 50.0
l = (4, 5, 6)       # tuple
list(l)             # list [4, 5, 6]
```

Argumenty při volání z terminálu

Když si chceme naprogramovat skript, ve kterém použijete některé argumenty zadané při volání z terminálu, budeme potřebovat standardní modul *sys*. Mějme skript *skript.py*.:

```
import sys
print(sys.argv[2])
print(sys.argv)
```

Pak při zavolání následujícím způsobem dostáváme output:

```
./skript.py testujeme 20 83 90.5
'20'
['./skript.py', 'testujeme', '20', '83', '90.5']
```

Zpracování a formátování textu

Textové řetězce neboli *stringy* si můžeme představit jako list znaků a platí pro ně stejné pravidla jako pro listy:

```
text = 'testovaci string'
text[4]                     # 'o'
text[:5]                    # 'testo'
text[2:5] + text[6:11]      # 'stovaci s'
```

Pro zpracování případně formátování stringů si zde uvedeme dvě užitečné funkce, z nichž první je *format*, která umožňuje výpis naformátovaného textu, který se hodí zejména při výpisu číselných hodnot. Zde jsou pouze její základní funkce, ale doporučujeme prostudovat dokumentaci pro další možnosti.
Funkce *format* umožňuje výpis naformátovaných stringů, které se hodí zejména při výpisu číselných hodnot.

```
'Vypocet: {} + {} se rovna {}'.format(1, 2, 3)   # 'Vypocet 1 + 2 se rovna 3'
'Vypocet: {0} + {0} se rovna {1}'.format(1, 2)   # 'Vypocet 1 + 1 se rovna 2'
                                                 #  - opakované použití proměnné
```

Formátování reálných čísel **{A:BC.DF}**, kde:

A číslo proměnné
B zarovnání – možnosti: vlevo <, střed ^, vpravo >
C šířka pole - celkový počet znaků, které se s proměnnou vypíší
D počet desetinných míst pro reálne čísla
F specifikace formátu – *f* pro reálné a *d* pro celé čísla

```
'Ukazka: {0:>10.5f}'.format(1.24)     # 'Ukazka:    1.24000'
'Ukazka: {0:>8.1f}'.format(1.24)      # 'Ukazka:      1.2'
'Ukazka: {0:<8.2f}'.format(1.24)      # 'Ukazka: 1.24    '
```

Jestli naopak potřebujeme rozdělit textový řetězec na jednotlivé části, používáme funkci *split*. Ta jako výchozí znak pro oddělování textu používá mezeru, ale je možné zadat jiný znak jako argument (např. *split(',')*). Příklad rozdělení stringu na list celých čísel:

```
inp = '21 232 43 24 5'          # string
inp.split()[2]                  # '43'
l = inp.split()                 # list ['21', '232', '43', '24', '5']
out = [int(i) for i in l]       # list [21, 232, 43, 24, 5]
```

Tyto operace můžeme pro stručnější zápis spojit do jedné řádky:

```
out = [int(i) for i in inp.split()]
```

Práce s maticemi

Pro práci s maticemi budeme používat již zmíněnou knihovnu *NumPy*. Nejprve uvádíme základy práce s maticemi a různé způsoby jejich inicializace.

```
import numpy as np
m = np.array([[2, 3, 4],
              [1, 3, 5]])
m.shape                         # vrací tvar matice jako tuple (2, 3)
m[1, 2]                         # vrací 5
m[0, 2] = 10                    # přiřadí číslo 10 prvku 0,1 tj. místo čísla 4
np.transpose(m)                 # vrací transponovanou matici
np.zeros((3,4))                 # vrací matici 3 krát 4 vyplněnou nulami
np.ones((3,4))                  # vrací matici 3 krát 4 vyplněnou jedničkami
np.diag(np.array([1,2,3]))      # vrací matici 3 krát 3 s diagonálou 1, 2, 3
```

Dále je možné matice sčítat a odčítat, provádět operace na všech prvcích, násobit matice po prvcích * (tj. *ij*-tý prvek s *ij*-tým) anebo standardním maticovým součinem @.

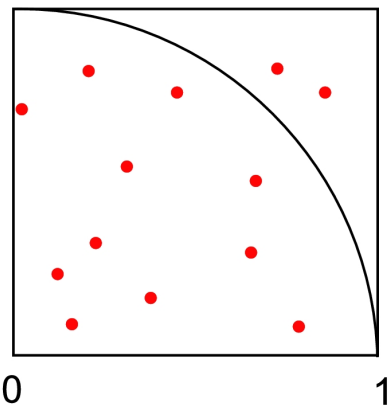
```
a = np.array([[1, 2],
              [3, 4]])
b = np.array([[2, 1],
              [6, 3]])

# zde uvádíme operaci a pod ní výslednou matici
# mocnina | součet | násobení po prvcích | maticové násobení | nerovnost
   a**2       a+b            a*b                  a@b            a<4
[1,  4]    [3, 3]       [ 2,  2]             [14,  7]      [True,  True]
[9, 16]    [9, 7]       [18, 12]             [30, 15]      [True, False]
```

2.11 Úlohy

1. Výpočet čísla π metodou Monte Carlo

   Napište program, který bude pomocí generátoru náhodných čísel počítat číslo π. Generujte body se souřadnicemi $x \in \langle 0, 1)$, $y \in \langle 0, 1)$. Počet bodů N_1, které padly do čtvrtiny jednotkového kruhu ($x^2 + y^2 < 1$), ku celkovému počtu generovaných bodů N odpovídá poměru ploch

   $$\frac{N_1}{N} = \frac{\pi}{4}. \qquad (2.1)$$

   Náhodné číslo v intervalu $\langle 0, 1)$ vygenerujete pomocí funkce `random()` z modulu `random`.

2. Napište funkci pro násobení matic podle vzorce

   $$C_{ij} = \sum_k A_{ik} B_{kj}. \qquad (2.2)$$

   Matice reprezentujte dvourozmernými poli (list listů). Funkci vyzkoušejte na malých, ručně naplněných maticích. Jaké je škálování násobení matic?

3 Program pro výpočty pH pufrů

3.1 pH pufru slabé kyseliny a silné báze

Pro případ pufru silné báze (B^+) a slabé kyseliny (HA) charakterizované disociací

$$\mathrm{HA} + \mathrm{H_2O} \rightleftarrows \mathrm{A^-} + \mathrm{H_3O^+}$$

platí následující čtyři rovnice:

$$[\mathrm{H_3O^+}][\mathrm{OH^-}] = K_\mathrm{W} = 10^{-14} \tag{3.1}$$

$$K_\mathrm{A} = \frac{[\mathrm{A^-}][\mathrm{H_3O^+}]}{[\mathrm{HA}]} \tag{3.2}$$

$$c_\mathrm{HA} = [\mathrm{A^-}] + [\mathrm{HA}] \tag{3.3}$$

$$[\mathrm{B^+}] + [\mathrm{H_3O^+}] = [\mathrm{OH^-}] + [\mathrm{A^-}] \qquad [\mathrm{B^+}] = c_\mathrm{B} \tag{3.4}$$

Rovnice (3.4) odpovídá rovnici elektroneutrality. Z těchto rovnic lze odvodit rovnici pro výpočet koncentrace hydroxoniových iontů

$$[\mathrm{H_3O^+}]^3 + (K_\mathrm{A} + c_\mathrm{B}) \cdot [\mathrm{H_3O^+}]^2 + (c_\mathrm{B}K_\mathrm{A} - K_\mathrm{W} - K_\mathrm{A}c_{HA}) \cdot [\mathrm{H_3O^+}] - K_\mathrm{A}K_\mathrm{W} = 0. \tag{3.5}$$

Její přesné řešení vyžaduje numerické metody (viz. dále). V praxi se proto pH pufru slabé kyseliny a silné báze obvykle počítá pomocí přibližného vzorce (Henderson-Hasselbalch):

$$\mathrm{pH} = \mathrm{p}K_\mathrm{A} + \log\frac{c_\mathrm{B}}{(c_\mathrm{HA} - c_\mathrm{B})} \tag{3.6}$$

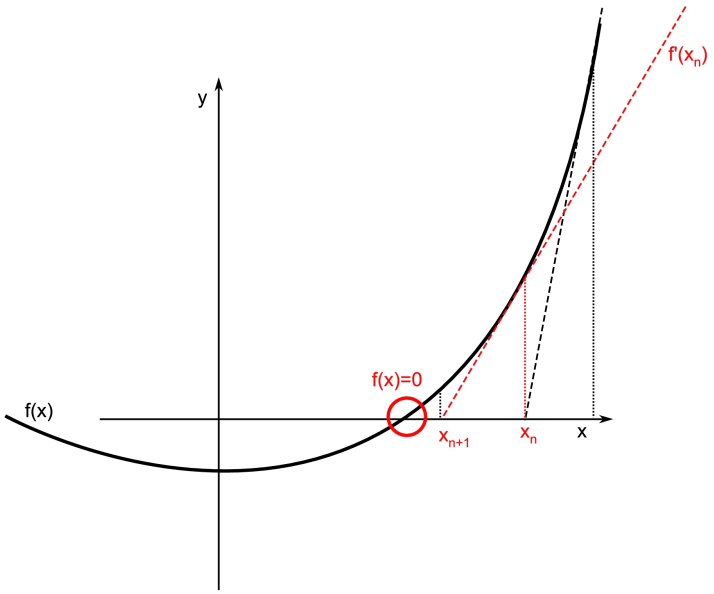
3.2 Newtonova iterační metoda

Kubickou rovnici (3.5) lze řešit numericky, *Newtonovou iterační metodou* (viz. obr. 3.1).
Pro následující krok x_{n+1} iteračního cyklu zřejmě platí

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{3.7}$$

3.3 Úlohy

1. Odvoďte rovnici (3.5).



Obrázek 3.1: Newtonova iterační metoda.

2. Napište program pro výpočet pH pufru slabé kyseliny a silné báze.
   Několik poznámek:

   - Použijte modul `math`, ze kterého budete potřebovat: `log10(a)` ($\log_{10} a$) a `fabs(a)` ($|a|$)
   - Vstupní parametry programu: $\mathrm{pK_A}$, c_{HA}, c_{B}
   - Použijte dostatečně malé kritérium konvergence pro $[H_3O^+]$, nezapomeňte, že pracujete s logaritmy.
   - Pro počáteční odhad $[H_3O^+]$ v Newtonově metodě použijte Hendersonův-Hasselbalchův vzorec (3.6).

3. Spočtěte pH směsi 50 ml roztoku kyseliny octové o koncentraci 0.1 M ($K_A = 1.8 \cdot 10^{-5}$) a 25 ml roztoku hydroxidu sodného o koncetraci 0.1 M (výsledek: pH = 4.7).

4. Napočítejte titrační křivku kyseliny octové (opakovaný výpočet pH pufru se zvyšující se koncentrací báze). Data vyneste do grafu pomocí programu Gnuplot.

5. Máte představu, kdy Hendersonův-Hasselbalchův vzorec (3.6) selhává? Ověřte výpočtem.

4 Úvod do programu *Gaussian*

Cílem tohoto cvičení je úvodní seznámení s programem *Gaussian* [3]. Během cvičení si prakticky vyzkoušíte různé *ab initio* metody, báze atomových orbitalů (AO), poznáte jejich přesnost i výpočetní náročnost.

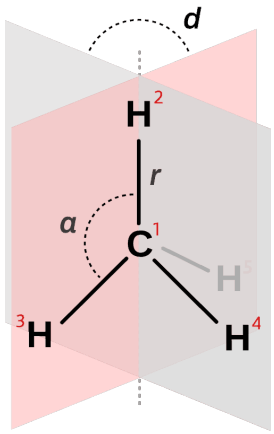
4.1 Z-matice

Geometrii molekuly je možné zadat dvojím způsobem:

1. **Kartézské souřadnice** – x, y, z pro každý atom ($3N$ parametrů)
2. **Vnitřní souřadnice (*Z-matice*)** – vazebné délky r, úhly a a dihedrální úhly d ($3N - 6$ parametrů)

Na Obrázku 4.1 je příklad Z-matice molekuly methanu spolu s její geometrií.

```
! Z-matice methanu
C
H 1 r
H 1 r 2 a
H 1 r 2 a 3  d
H 1 r 2 a 3 -d

r = 1.0
a = 109.5
d = 120.0
```



Obrázek 4.1: Z-matice molekuly methanu a schéma její geometrie.

Nejprve se udává typ atomu, následně délka vazby (v principu to nemusí být skutečná vazba, pouze vektor spojující nějaké atomy), úhlem a dihedrálním úhlem. Použije se vždy tolik parametrů, aby byla geometrie jednoznačná, např. pro druhý atom stačí pouze délka vazby. Před délkou vazby se udává číslo referenčního atomu od kterého se měří. Úhel se měří mezi přímkami z nichž jedna prochází zadávaným atomem a atomem, od kterého se měří vzdálenost, a druhá atomem, od kterého se měří vzdálenost, a atomem specifikovaným před daným úhlem. Dihedrální úhel svírají dvě roviny z nihž každá je definována třemi body – první je definována trojicí bodů použitých pro definici vazebného úhlu a druhá je předešlá trojice bodů s tím, že zadávaný atom se zamění za atom specifikovaný před dihedrálním úhlem.
Při stavbě molekul není možné použít úhel 180° (zamyslete se proč). Pokud je potřeba vytvořit lineární molekulu (např. acetylen) nebo pro zjednodušení Z-matic složitejších molekul (viz. dále), lze použít tzv. *dummy* atomy. Ty se označují písmenem **X** a slouží jako pomocné body, které se nezahrnují do výpočtů.

Úlohy

Sestavte a v programu *Molden* si zobrazte Z-matice následujících molekul:

- benzen (2 *dummy* atomy - jeden uprostřed kruhu, druhý nad ním)
- kyselina octová
- amoniak (1 *dummy* atom, 2 parametry + 1 proměnlivý pro inverzi)
- ethan v zákrytové konformaci (4 parametry + 1 dihedrální úhel pro rotaci kolem C–C)

4.2 Input programu *Gaussian*

Zde uvádíme příklad inputu pro výpočet energie molekuly vody v minimální bázi metodou HF. Je důležité si pamatovat, že všechny hodnoty parametrů musí být typu float, tj. obsahovat desetinnou čárku a na konci inputu ponechávame prázdný řádek. V případě, že chceme víc detailů o výpočtu, začínáme řádek s klíčovými slovy **#P**
Program se pak spustí příkazem **G16 input_file 1**

```
# HF/STO-3G                    ! klíčová slova: metoda/báze atd.

voda v minimalni bazi          ! název výpočtu, komentář

0 1                            ! náboj, multiplicita
O                              ! Z-matice
H 1 r
H 1 r 2 a

r = 1.0                        ! parametry
a = 104.5
                               ! prázdný řádek !!!
```

4.3 Geometrické optimalizace malých molekul

V této úloze si vyzkoušíte zoptimalizovat geometrii několika malých molekul pomocí metod různé úrovně a s různě velkými bázemi.

Postup

1. V první řádce inputu zvolíme geometrickpu optimalizaci přidaním klíčového slova **Opt**
2. Vyberte si dvě z následujících molekul: H_2, CO, H_2O, H_2O_2, NH_3.
3. Zoptimalizujte jejich geometrie na následujících úrovních:
   HF/STO-3G, HF/6-31G*, B3LYP/6-31G*, MP2/6-31G*, CCSD/cc-pVDZ a CCSD/cc-pVTZ
4. Pro molekulu H_2O (HF/6-31G*) přidejte ke klíčovým slovům **Pop=Full GFInput**, což umožní výpis výsledných molekulových orbitalů.

Výsledky

1. Porovnejte výsledné struktury s experimentálními údaji (Tabulka 4.1).
2. Porovnejte časovou náročnost jednotlivých metod a bází.
3. Podívejte se na molekulové orbitaly H_2O, otevřením outputu v programu *Molden* (budete instruováni).

Tabulka 4.1: Experimentální struktury zmíněných malých molekul.

H_2	$r = 0.74144$ Å	H_2O	$r = 0.9578$ Å	H_2O_2	$r_{\text{O-H}} = 0.967$ Å
			$a = 104.48°$		$r_{\text{O-O}} = 1.4556$ Å
CO	$r = 1.12832$ Å	NH_3	$r_{\text{N-H}} = 1.016$ Å		$a_{\text{O-O-H}} = 102.32°$
			$a_{\text{H-N-H}} = 106.7°$		$a_{\text{dih}} = 113.70°$

Poznámky

- Pozor na symetrii výchozí geometrie – v případě vyšší symetrie, než má skutečná molekula, ji bude program držet v průběhu celé optimalizace. Toto chování možno obejít použitím klíčového slova **NoSym**.

4.4 Pyramidální inverze amoniaku

Postup

1. Proveďte geometrickou optimalizaci planární a pyramidální struktury molekuly NH_3 metodou **HF** s následujícími bázemi: **STO-3G**, **6-31G**, **6-31G***, **6-31G****.
2. Do přehledné tabulky zapište hodnoty energetických bariér pyramidální inverze pro různé báze.
3. Na zoptimalizovaných geometriích (např. s bází **6-31G**) proveďte single point energetické výpočty s klíčovým slovem **Pop=Full** (vypíše podrobonou populační analýzu).

Výsledky

1. Popište trend, porovnejte hodnoty s experimentálním údajem 5.8 kcal/mol.
2. Porovnejte energie obsazených orbitalů obou molekul.

Poznámky

- Pro převod energií z a.u. do kcal/mol je potřeba energie vynásobit koeficientem 627.5.

4.5 Rotační bariéra ethanu

Postup

1. Požijte Z-matici pro ethan v zákrytové konformaci tak, aby byla připravena na přechod k nezákrytové konformaci variováním jednoho dihedrálního úhlu (viz úkoly v sekci 4.1).

2. Strukturu zoptimalizujte na úrovni **HF/6-31G**.
3. Vemte zoptimalizovanou geometrii a proveďte tzv. scan: ke klíčovým slovům přidejte **Scan** a u definice dihedrálního úhlu použijte **d = 0.0 36 10.0**. Výsledkem je série výpočtů pro hodnoty dihedrálního úhlu $0° - 360°$ (po $10°$).

Výsledky

1. Tabulku hodnot (dihedrální úhel, energie) zkopírujte do nového souboru pojmenovaného např. *data*.
2. Hodnoty vyneste do grafu v programu *Gnuplot* příkazem **plot "data" smooth csplines**
3. Hodnotu rotační bariéry porovnejte s experimentální hodnotou: 2.9 kcal/mol.

4.6 Disociace molukuly vodíku v minimální bázi

Postup

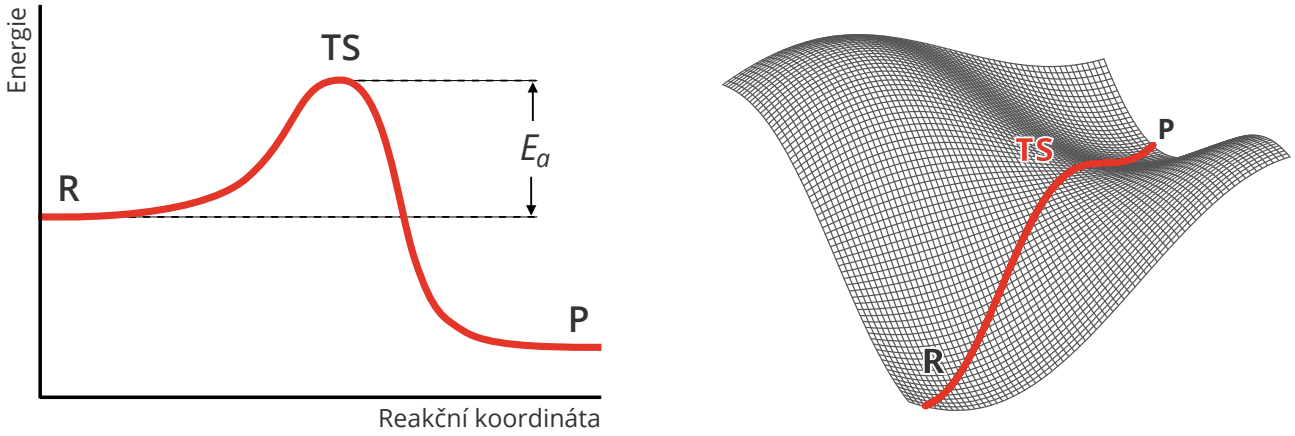
1. Proveďte výpočet disociace molekuly H_2 metodami **RHF** v bázi **STO-3G**: klíčové slovo **Scan**, proměnná **r = 0.3 27 0.1**
2. Proveďte scan **r = 3.0 27 -0.1** metodou **UHF** a tentokrát do klíčových slov přidejte **Guess=Mix NoSymm**.
3. Spočtěte disociaci metodami **CID**, **MP2** a **CISD**.

Výsledky

1. Získané křivky (**RHF**, **UHF**, **CID**, **MP2** a **CISD**) vyneste do jednoho grafu v programu *Gnuplot*. Je mezi nimi rozdíl? Napadne Vás proč?
2. Proč selhává metoda **RHF** při disociaci molekuly vodíku?

5 Tranzitní stavy

Cílem tohoto cvičení je geometrická optimalizace systémů s cílem nalezení jejich tranzitních stavů (TS). Jedná se o stav, ve kterém má studovaný systém nejvyšší energii na potenciálovém povrchu podél reakční koordináty. Jestliže vememe do úvahy také ostatní stupně volnosti, jedná se o sedlový bod tj. bod na potenciálovém povrchu, který je minimem ve všech směrech kromě jednoho (reakční koordináty), viz Obrázek 5.1.



Obrázek 5.1: Energetický průběh reakce podél reakční koordináty (vlevo) a pohyb po reakční koordinátě znázorněný na potenciálovém povrchu (vpravo). Značení: reaktantany R, produkty P a tranzitní stav TS.

Správné stanovení tranzitních stavů umožňuje studium termodynamiky chemických reakcí. Optimalizací geometrie reaktantů a tranzitních stavů umíme například stanovit výšku potenciálové bariéry a tedy i aktivační energie E_a. Zní pak můžeme určit rychlost reakce pomocí Arrheniovy rovnice

$$k = Ae^{-\frac{E_a}{RT}}, \tag{5.1}$$

kde A je frekvenční konstanta, R plynová konstanta a T teplota.

Jelikož je hledání sedlových bodů mnohem náročnější než hledání energetických minim, omezíme se jenom na jednoduché reakce – nukleofilní substitucí halometanů a izomerizaci 2-pyridonu – na kterých si ukážeme dva různé přístupy geometrické optimalizace tranzitních stavů.

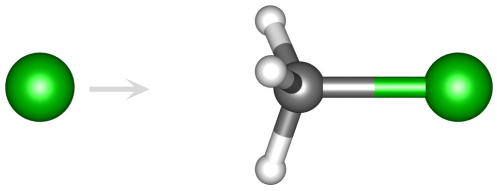
Všechny výpočty budeme provádět metodou DFT s funkcionálem **B3LYP** a v bázi **cc-pVDZ**.

5.1 S_N2 nukleofilní substituce halometanů

Při S_N2 nukleofilních substitucích dochází k simultánnímu navázání nukleofilní a odstoupení odcházející skupiny. Jednoduchým případem takovýchto reakcí jsou substituce halometanů jako například

$$Cl^- + CH_3Cl \longrightarrow CH_3Cl + Cl^- \tag{5.2}$$

znázorněná taky na Obrázku 5.2. V nasledující úloze budeme zkoumat tuto reakci pro tři nejlehčí halogeny: fluor, chlor a brom.



Obrázek 5.2: Nukleofilní substituce chlorometanu.

Postup optimalizace TS a scan podél reakční koordináty

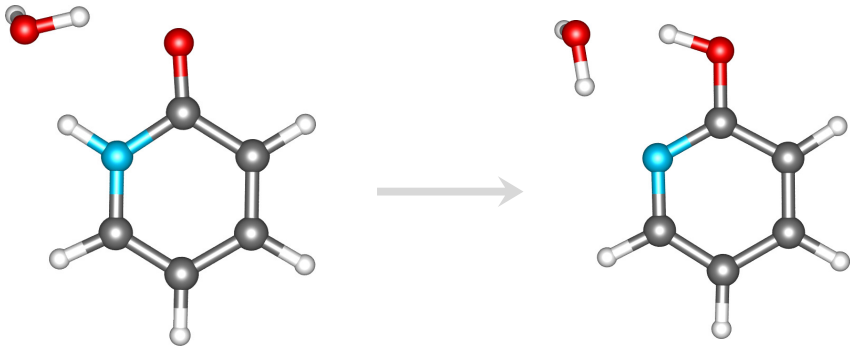
1. V *Moldenu* vymodelujte systém v tranzitním stavu. Pro jeho úspěšné nalezení je důležitá správná počateční geometrie, která musí být blízko finální geometrii!
2. Proveďte optimalizaci pomoci klíčového slova **Opt=(TS,CalcFC)** a následní vibrační analýzu s **Freq** (můžete zadat obě klíčová slova najednou). Nezapomeňte nastavit správný náboj pro výpočet.
3. Zobrazte si spektra normální módy v *Moldenu* (otevřete output a kliknite na *Norm. Mode*). V případě, že výpočet skonvergoval správně do TS měli byste pozorovat imaginární frekvenci.
4. Pro scan potencialového povrchu podél reakční koordináty vložte do inputu geometrii zoptimalizovanou pro TS a přidejte klíčová slova **IRC=(CalcFC,MaxPoints=30,ReCorrect=Never)**.

Úkoly

1. Proveďte optimalizace jak tranzitního tak koncového stavu symetrických substitucí (5.2) F, Cl, Br. Spočtěte aktivační energie. Která reakce je nejrychlejší a kolikrát pomalejší jsou vůči ní ostatní reakce při pokojové teplote? Využijte (5.1) a předpokládejte, že faktor A je stejný pro všechny substituenty.
2. Zoptimalizujte tranzitní stav jedné symetrické a jedne asymetrické substituce a proveďte scan potenciálového povrchu podél reakční koordináty. Křivky vyneste do grafu.
3. Jakému pohybu odpovídá imaginarní frekvence pro systém v tranzitním stavu?

5.2 Tautomerie 2-pyridonu

Často se stává zejména u složitějších systémů, že neumíme odhadnout počáteční geometrii, ze které by program dokázal skonvergovat do tranzitního stavu. V této úloze si na příkladě přesunu protonu v molekule 2-pyridonu ukážeme, jak je najít pomocí algoritmu, který využívá znalost geometrie reaktantů a produktů. Budeme simulovat jak přímou reakci, tak v přítomnosti molekuly vody v místě přesunu (viz Obr. 5.3).



Obrázek 5.3: Přesun protonu v 2-pyridonu za přítomnosti molekuly vody.

Postup

1. V *Moldenu* vymodelujte molekulu 2-pyridonu a to jak reaktantu, tak produktu (viz Obr. 5.3) s a bez molekuly vody v míste reakce (budete mít 4 geometrie). Struktury zoptimalizujte.
2. Pro hledání tranzitního stavu použijte klíčová slova **Opt=(QST2,CalcFC)** a do inputu pod sebe vložte zoptimalizované geometrie reaktantu a produktu, spolu s řádkou s komentářem, nábojem a multiplicitou.
3. Ověřte, že nalezené struktury jsou opravdu v tranzitním stavu.
4. Proveďte scan potencialového povrchu podél reakční koordináty s klíčovými slovy (viz předchozí úloha), kde nastavíte **StepSize=15,MaxPoints=20**.

Úkoly

1. Spočtěte reakční a aktivační entalpie, Gibbsovy energie (298 K) a vyzkoušejte si výpočet rovnovážné konstanty a rychlostních konstant.
2. Vykreslete potenciálové křivky podél reakční koordináty. Jak se liší?

6 Mezimolekulové interakce

Vrámci tohoto cvičení si vyzkoušíte výpočet interakční energie slabých mezimolekulových sil a řešení problémů spojených s použitím konečné, a tedy neúplné báze. V prvním cvičení spočteme všechny korekce ručne abychom si vysvětlili princip použitých přístupů, v druhém si pak ukážeme poněkud praktickejší přístup implementovaný přímo v *Gaussianu*.

Kdybychom chtěli provést výpočet přesné nerelativistické energie, musely by být splňené dvě podmínky. Prvně by bylo nutné provést přesný výpočet zahrnující korelační energii definovanou jako

$$E^{\text{corr}} = E^{\text{exact}} - E^{\text{HF}}, \tag{6.1}$$

tedy jako rozdíl mezi přesnou nerelativistickou energií E^{exact} a Hartree-Fockovou energií E^{HF}. Metodou poskytující přesnou energii je například FCI. Tato metoda je však v praxi použitelná jen pro velmi malé systémy a proto se obvykle provádějí jen výpočty s přibližními metodami, kterými se snažíme zahrnout co nejvíc korelační energie např. CCSD (tj. CC s omezením na jedno- a dvouexcitace). Za druhé provádíme výpočty v konečné a tedy neúplné bázi. I když je častokrát postačující použít dostatečně velkou bázi, ale opravdu přesné výpočty je potřeba provádět v tzv. *limitě úplné báze* (CBS - *complete basis set*).

6.1 Limita úplné báze

Jednou z možností jak získat energie v CBS limitě, je použití Dunningových *korelačně konzistentních bází*. Je to sada postupně zvětšujících se bází, navržena tak, aby hladce a rychle konvergovala k CBS limitě. Jsou označeny jako cc-pVζZ, kde písmeno $\zeta = \text{D, T, Q, 5, 6}\ldots$ označuje velikost báze (DZ – double zeta, TZ – triple zeta,...). Provedením výpočtů v nejméně třech různých bázích můžeme pak získané HF energie nafitovat funkcí

$$E_\zeta^{\text{HF}} = E_\infty^{\text{HF}} + Ae^{-B\zeta}, \tag{6.2}$$

kde hledanými parametry jsou A, B a především E_∞^{HF}, který představuje právě hledanou HF energii v CBS limitě. Pro extrapolaci korelační energie je zas možné použít vztah

$$E_\zeta^{\text{corr}} = E_\infty^{\text{corr}} + A\zeta^{-3}. \tag{6.3}$$

V případě bází označených písmeny pak tyto nahradíme odpovídajícími čísly tj. DZ $= 2$ atd.

6.2 Interakční energie

Jelikož si v tomto cvičení ukážeme výpočet interakčních energií dimérů, omezíme se v následujícím textu na popis dvou slabě interagujících systémů, ale stejné principy samozřejmě platí i pro větší klastry. In-

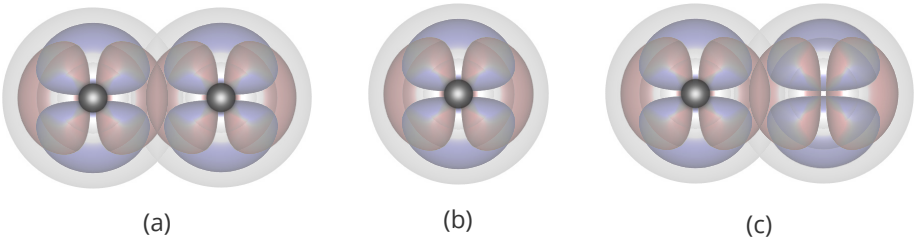
terakční energii mezi dvěma subsystémy A a B složeného systému AB můžeme spočíst jako

$$E_{\text{int}} = E_{\text{AB}} - E_{\text{A}} - E_{\text{B}}. \tag{6.4}$$

Při takovémto přístupu však narazíme na problém označovaný jako *basis set superposition error* (BSSE), který má za následek přílišnou stabilizaci diméru. Tento efekt vzniká důsledkem neúplnosti bází a je tedy výrazný zejména u výpočtů v menších bázích. Při výpočtu diméru AB totiž monomér A využíva pro popis své elektronové struktury částečně také bázové funkce lokalizované na monoméru B a naopak.
Jedním z řešení je použití *counterpoise korekce* (CP), kdy se výpočet monoméru A provádí za přítomnosti báze lokalizované na atomech monoméru B. Tyto tzv. *ghost atomy* však nemají žádný náboj na jádře ani žádné elektrony. Interakční energii s CP korekcí pak můžeme vyjádřit jako

$$E_{\text{int}}^{\text{CP}} = E_{\text{AB}}^{\text{AB}}(\text{AB}) - E_{\text{A}}^{\text{AB}}(\text{AB}) - E_{\text{B}}^{\text{AB}}(\text{AB}), \tag{6.5}$$

kde $E_{\text{X}}^{\text{Y}}(\text{Z})$ značí energii fragmentu X v geometrii optimalizované pro fragment Y v bázi fragmentu Z. Například $E_{\text{A}}^{\text{AB}}(\text{AB})$ znamená energii, kde zoptimalizujeme geometrii systému AB a pak provedeme výpočet energie pro subsystém A v této geometrii, přičemž při výpočtu použijeme také bázi subsystému B v podobě *ghost atomů*. Tento přístup je znázorněn na Obrázku 6.1.



Obrázek 6.1: Znázornění výpočtů pro výpočet interakční energie diméru helia v DZ bázi: (a) dimér $E_{\text{AB}}^{\text{AB}}(\text{AB})$, (b) monomér $E_{\text{A}}^{\text{A}}(\text{A})$, (c) monomér s CP korekcí v podobě *ghost atomu* $E_{\text{A}}^{\text{AB}}(\text{AB})$.

Při takto provedeném výpočtu se však nezohlední energetický rozdíl, vzniklý v důsledku rozdílných geometrií izolovaných monomérních jednotek a jejich geometrií v diméru. Vazebnou energii zohledňující tyto deformace můžeme vyjádřit

$$E_{\text{bind}} = E_{\text{AB}}^{\text{AB}}(\text{AB}) - E_{\text{A}}^{\text{A}}(\text{A}) - E_{\text{B}}^{\text{B}}(\text{B}) - \Big[E_{\text{A}}^{\text{AB}}(\text{AB}) - E_{\text{A}}^{\text{AB}}(\text{A}) + E_{\text{B}}^{\text{AB}}(\text{AB}) - E_{\text{B}}^{\text{AB}}(\text{B})\Big]. \tag{6.6}$$

6.3 Výpočet interakční energie diméru vody

Postup

1. V *Moldenu* vymodelujte dimér vody a strukturu zoptimalizujte na úrovni **CCSD/cc-pVDZ**.
2. V získané gemetrii napočítejte single-point (SP) energie (vynecháním klíčového slova **Opt**) pro monoméry 1 a 2 – celkem 4 výpočty:
   - bez CP: Umažte druhou molekulu.
   - s CP: K atomům druhé molekuly přidejte **-bq**, čímž se z nich stanou *ghost atomy*: např. *O-bq*

3. Výpočty z předchozího kroku zopakujte v bázích **cc-pVTZ** a **cc-pVQZ**. V těchto bázích proveďte taky SP výpočty diméru.
4. Hodnoty vyneste do tabulky.

Úkoly

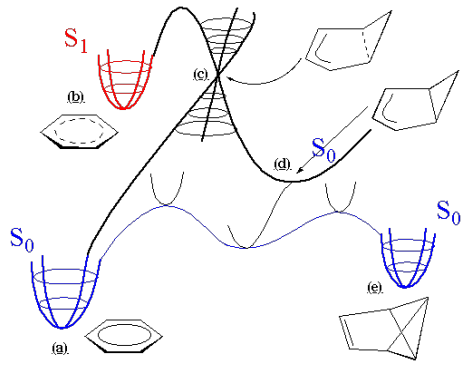
1. Extrapolujte separátně HF a korelační energie dle (6.2) a (6.3).
2. Spočtěte interakční energie s použitím a bez použití CP korekce. Jaká velká je BSSE a jak se mění vplyv CP korekce se zvětšující se bází?
3. Zběhnete výpočty pro zbylé dva členy v **cc-pVDZ** (jeden výpočet) a spočtěte interakční energii dle (6.6). Jak velký je přispěvek deformační energie $E_{\text{def}} = E_{\text{bind}} - E_{\text{int}}$.

7 Excitované stavy

Náplní tohoto cvičení je úvod do kvantové chemie excitovaných stavů. Nejprve se budeme zabývat fotochemickým přesmykem benzenu na benzvalen [6], kde si vyzkoušíte optimalizaci kónické intersekce. V druhé části cvičení budete počítat elektronická spektra formaldehydu.

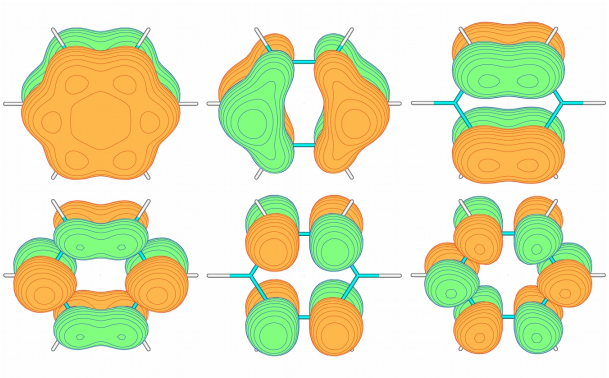
7.1 Fotochemická izomerizace benzenu na benzvalen

V této úloze budeme používat výstup z jednoho výpočtu jako vstup pro druhý výpočet. K tomuto účelu slouží tzv. *checkpoint* soubory - pro program čitelné binární soubory s výsledky výpočtů. Specifikace klíčových slov se přesune na druhou řádku, první řádka musí obsahovat **%chk=název_souboru** (obvykle s koncovkou *.chk*). Protože v případě načítání výsledků z *checkpoint* souborů je *Gaussian* následně přepíše, je dobré si dělat zálohy (např. použít samostatný adresář pro každý výpočet).
Pro tuto úlohu budeme všechny výpočty provádět s klíčovým slovem **NoSymm** pro vypnutí symetrie.



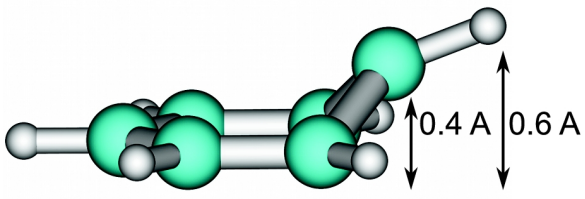
Postup

1. V *Moldenu* si vytvořte molekulu benzenu a uložte ve formátu xyz (jednotky jsou Å) a pro tuto geometrii spočtěte **HF/STO-3G**, s klíčovými slovy pro zobrazení molekulových orbitalů **Pop=Full GFInput**.
2. Aktivní prostor pro metodu CASSCF sestavíme z molekulových orbitalů vzniklých z p_z atomových orbitalů uhlíků, viz Obrázek 7.1. *Moldenem* si zobrazte molekulové orbitaly a najděte ty, které jim odpovídají.
3. Zoptimalizujte geometrii pomocí **CASSCF(6,6)/STO-3G**, použijte **Guess=(Read,Alter)**. Počáteční odhad molekulových orbitalů se vezme z *checkpoint* souboru a díky **Alter** můžeme přeuspořádat počáteční molekulové orbitaly tak, aby orbitaly pro aktivní prostor byly seřazeny za sebou – za geometrií následuje prázdný řádek a pak dvojice čísel vyjadřující, které orbitaly se mají prohodit.



Obrázek 7.1: Orbitaly pro CASSCF(6,6) výpočet.

4. Zoptimalizovanou geometrii načtěte pomocí **Guess=Read Geom=Check** (už nevypisujte souřadnice do inputu), udělejte single-point výpočet v **4-31G** bázi. Následně krok zopakujte avšak načtěte orbitaly z **4-31G** báze pro výpočet v **6-31G*** bázi.
5. Proveďte geometrickou optimalizaci v **6-31G*** bázi. Proč jsme se asi vydali takovouto oklikou?
6. Spočtěte vertikální excitační energii. Energii n-tého stavu získáte pomocí **CASSCF(6,6,NRoot=*n*)**.
7. Vypočítejte adiabatickou excitační energii – najděte stacionární bod na povrchu potenciální energie excitovaného stavu (S_1) a ověřte, že se jedná o energetické minimum.
8. V případě hledání kónické intersekce se z praktických důvodů omezíme na **STO-3G** bázi. Protože její geometrii předpokládáme neplanární, musíme lehce upravit počáteční geometrii, viz Obrázek 7.2. Pro výpočet pak použijte **CASSCF(6,6,NRoot=2,NoCPMCSCF) Opt=Conical**. Zoptimalizovanou geometrii si zobrazte *Moldenem*.



Obrázek 7.2: Výchozí geometrie pro optimalizaci kónické intersekce

7.2 UV spektrum formaldehydu

Nejjednodušší metodou výpočtu excitovaných stavů je konfigurační interakce se single-excitacemi (CIS). Jedná se o rychlou metodu, která je jednoduchá na použití (typicky stačí nastavit počet exitovaných stavů) a umožňuje charakterizovat elektronické přechody jako jednoelektronové: $\pi \rightarrow \pi^*$. Nevýhodou je však její omezená přesnost, a proto se dá prakticky použít pouze v případě přechodů s dominantním příspěvkem single-excitacíi (jinak se chyby pohybují v řádu eV).

Postup

1. Zoptimalizujte molekulu formaldehydu pomocí **CIS(NStates=5)/6-31G*** s klíčovými slovy pro vykreslení molekulových orbitalů.
2. Dále zoptimalizujte tuto molekulu pomocí **B3LYP/6-31G*** (podle možností můžete také v bázi cc-pVTZ) a udělejte TDDFT výpočet použitím klíčového slova **TD(Nstates=5)**.
3. Excitační energie $^1A_2$ a $^1B_2$ stavů porovnejte s experimentálními údaji, viz Tabulka 7.1.
4. V *Moldenu* si zobrazte molekulové orbitaly z CIS výpočtu a určete, jakými přechody jsou tyto excitované stavy charakterizovány.

Tabulka 7.1: Experimentální excitační energie formaldehydu.

$^1A_2$	4.07 eV
$^1B_2$	7.11 eV

8 Implementace molekulové dynamiky

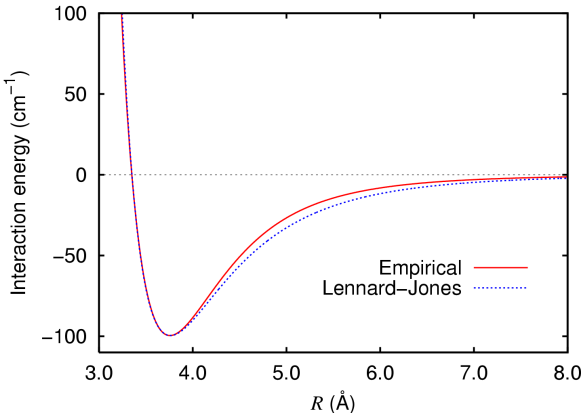
V této kapitole si naprogramujete Verletův integrátor, s jehož pomocí napíšeme program pro jednoduchou molekulovou dynamiku atomů argonu, jejichž interakci budeme modelovat pomocí Lennard-Jonesůvho potenciálu.

8.1 Lennard-Jonesův potenciál

Jako jednoduchý model interakce mezi neutrálními molekulami (atomy) se používá Lennard-Jonesův potenciál

$$V_{\mathrm{LJ}}(r) = 4\varepsilon\left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right), \tag{8.1}$$

který závisí pouze na dvou parametrech. Hloubka potenciálové jámy je dána parametrem ε a σ označuje vzdálenost pro kterou je potenciál rovný nule. Takovýhle tvar potenciálu (viz Obrázek 8.1) zajišťuje, že mezimolekulová interakce je silně odpudivá pro malé vzdálenosti, avšak mírně přitažlivá pro velké vzdálenosti.



Obrázek 8.1: Lennard-Jonesův potenciál pro dimer argonu ve srovnání s empirickým potenciálem.

8.2 Verletova integrační metoda

Klasická molekulová dynamika je řízena zákony klasické mechaniky, tj. Newtonovými pohybovými rovnicemi

$$\mathbf{f}_i = m_i \frac{\mathrm{d}^2\mathbf{r}_i(t)}{\mathrm{d}t^2} = -\nabla V(\mathbf{r}_i), \tag{8.2}$$

kde $\mathbf{f}_i$ je síla působící na i-tý atom, $\mathbf{r}_i$ jeho poloha, m_i hmotnost a $V(\mathbf{r}_i)$ je skalární potenciál v místě atomu.

Jako přímočaré řešení se nabízí prostý rozvoj

$$\mathbf{r}_i(t+\delta t) = \mathbf{r}_i(t) + \mathbf{v}_i(t)\delta t + \frac{1}{2}\mathbf{a}_i(t)\delta t^2, \tag{8.3}$$

kde δt označuje časový krok, $\mathbf{v}(t)$ rychlost atomu a $\mathbf{a}_i = \frac{\mathbf{f}_i}{m_i}$ jeho zrychlení. Takové řešení je však ireversibilní, jelikož předpokládá, že se rychlosti ani zrychlení během časového kroku δt nemění, což by však i pro malé hodnoty časového kroku vedlo k velké kumulaci chyb. Tento nedostatek je však možné jednoduše odstranit – symetrizací integrátoru dostáváme Verletův vztah pro polohu atomů

$$\mathbf{r}_i(t+\delta t) = 2\mathbf{r}_i(t) - \mathbf{r}_i(t-\delta t) + \mathbf{a}_i(t)\delta t^2, \tag{8.4}$$

a taky jejich rychlostí

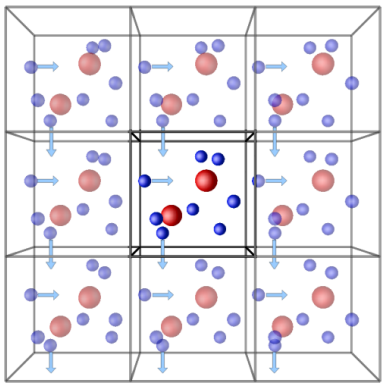
$$\mathbf{v}_i(t) = \frac{\mathbf{r}_i(t+\delta t) - \mathbf{r}_i(t-\delta t)}{2\delta t}. \tag{8.5}$$

Pokud však potřebujeme integrátor, který explicitně závisí na rychlostech, můžeme použít jeho alternativní formulaci zvanou *velocity Verlet*. V tomto případě je vztah pro polohu identický s (8.3), přičemž změnu rychlosti je možné spočíst jako

$$\mathbf{v}_i(t+\delta t) = \mathbf{v}_i(t) + \frac{\mathbf{a}_i(t) + \mathbf{a}_i(t+\delta t)}{2}\delta t. \tag{8.6}$$

8.3 Periodické okrajové podmínky

Aby bylo možné studovat systémy kondenzované fáze, případně roztoků bez vlivu povrchových jevů, zavádí se aproximacei, která se nazývá periodické okrajové podmínky (PBC z anglického *periodic boundary conditions*) (viz Obrázek 8.2). Systém který je ještě možné spočíst se uzavře do krychle, která je pak nakopírovaná do všech okolních stran. V případě, že atom opustí centrální krychli, objeví se následně na protilehlé straně v tom samém místě.
Abychom předešli neúměrnému množství interakcí, zavádí se dále konvence minimálního obrazu. To znamená, že v danou chvíli interaguje jeden atom právě s jednou kopií ostatních atomů a to s tou, která je mu nejblíž.



Obrázek 8.2: Periodické okrajové podmínky.

8.4 Berendsenův termostat

Pokud bychom implementovali molekulovou dynamiku jenom na základě dosud uvedených ..., při běhu by byl generován mikrokanonický NVE soubor. Jestli chceme simulovat kanonický NVT soubor, je potřeba zavést kontrolu teploty. Tu je možné pro systém pozostávající z N atomů definovat jako

$$\frac{3}{2}Nk_{\mathrm{B}}T = \frac{1}{2}\sum_i m_i v_i^2, \tag{8.7}$$

kde k_{B} je Boltzmannova konstanta, m_i hmotnost a v_i rychlost i-tého atomu.
Jednou z nejjednodušších možností je použít Berendsenův termostat, jenž přeškálovávaním rychlostí simuluje intereakci systému s tepelnou lázní. Takový rychlostní škálovací faktor má tvar

$$\lambda = \sqrt{1 + \frac{\delta t}{\tau}\left(\frac{T}{T_0} - 1\right)}, \tag{8.8}$$

kde je T aktuální teplota systému, T_0 teplota lázně a τ určuje intenzitu výměny tepla s tepelnou lázní (čim menší hodnota, tím větší vazba s lázní). Pro použití termostatu potřebujeme integrátor s explicitní závislostí poloh na rychlostech.

8.5 Než začnete

1. Ukažte, že Lennard-Jonesův potenciál má minimum v $r_{\mathrm{min}} = \sqrt[6]{2}\sigma$ a jeho hodnota je $V_{\mathrm{LJ}}(r_{\mathrm{min}}) = -\varepsilon$.
2. Spočtěte derivaci $\frac{\partial V_{\mathrm{LJ}}(r)}{\partial x}$. Pro zjednodušení nezapomeňte na *chain rule*. Jak vypadá gradient $\nabla V_{\mathrm{LJ}}(r)$?
3. Podobně jako v (8.3) rozviňte vztah pro polohu $\mathbf{r}_i(t - \delta t)$ a odvoďte Verletův integrátor (8.4) a (8.5).

8.6 Struktura programu

V programu budeme používat jednotek $\sigma = \varepsilon = m_{\mathrm{Ar}} = 1$. V těchto jednotkách je pak jednotka času rovná $\tau = \sqrt{\frac{m\sigma^2}{\varepsilon}}$.
Doporučujeme rozdělit si práci na několik samostatních funkcí:

```
def gradient_Lennard_Jones(r):                  # vrací vektor gradientu LJ potenciálu
def initialize(inp, box_size, sgm, eps, mass):  # inicializuje výpočet a vrací list atomů
def calculate_forces(atoms):                    # vynuluje a spočte síly na atomech
def velocity_verlet_step(atoms, dt):            # jeden krok pomocí Verletova integrátoru
```

Dále máte předpřipravenou třídu Atom pro jednotlivé atomy. Potřebujete pouze dopsat funkce podle komentářů v jejich definici.

```
class Atom():
    def __init__(self, coordinates, velocities, box_size):
      '''Konstruktor.'''
        self.r = np.array(coord)          # poloha atomu r(t)
        self.v = np.array(velocities)     # rychlost atomu v(t)
        self.f = 0                        # síla působíci na atom
        self.r_prev = None                # poloha atomu r(t-dt)
        Atom.box_size = box_size          # velikost boxu, proměnná společná pro celou třídu

```

```
10    def set_new_coordinates(self, r):
11      '''Nastaví koordináty – pozor na PBC!'''
12
13    def nearest_neighbour(self, that):
14      '''Vrací vektor vzdálenosti "dr" k nejbližší kopii atomu "that".'''
15      return dr
16
17    def force_on(self, that):
18      '''Připočte sílu, kterou působí atom "that" na atom.'''
```

Program bude probíhat nasledovně:

1. Inicializace.
   - Načítání počátečních poloh a rychlostí ze souboru *init.py* pomocí **from init import ***.
   - Jelikož jsou polohy uvedeny v Ånsgtrömech rychlosti v m/s, je potřeba je převést do námi používaných jednotek tj. Å $\rightarrow \sigma$ a $\frac{m}{s} \rightarrow \frac{\sigma}{\tau}$.
   - Vytvoření instancí třídy Atom pro všechny atomy.
   - Spočtěte síly působící na všechny atomy.
2. Krok pomocí velocity Verlet integrátoru – opakujte pro zadaný počet kroků dlouhých $\delta t = 10^{-3}$.
   - Spočtěte nové polohy atomů $r(t + \delta t)$.
   - Spočtěte síly působící na všechny atomy.
   - Spočtěte rychlosti $v(t + \delta t)$.
3. Ukládejte si polohu v podobě *xyz* souboru každých 50 kroků.

9 Implementace metody RHF

9.1 Trocha teorie

Vyjděme z Hartree-Fockových rovnic (v kanonickém tvaru, s odintegrovanou spinovou proměnnou) [4]

$$\hat{f}\psi_i(\mathbf{r}_1) = \varepsilon_i\psi_i(\mathbf{r}_1), \tag{9.1}$$

kde $\hat{f}$ je nelokální Fockův operátor zahrnující jedno- a dvouelektronový (Coulombický a výměnný) příspěvek, ψ_i molekulové orbitaly a ε_i jejich energie. Jestliže uděláme MO-LCAO rozvoj

$$\psi_i = \sum_\mu c_{\mu i}\phi_\mu, \tag{9.2}$$

kde ϕ_μ jsou atomové orbitaly (pozn.: latinské indexy označují MO a řecké AO), můžeme vyjádřit HF rovnice v maticovém tvaru – Roothaanovy rovnice:

$$\mathbb{F}\mathbb{C} = \mathbb{S}\mathbb{C}\varepsilon, \tag{9.3}$$

kde $\mathbb{F}$ je Fockova matice ($F_{\mu\nu} = \langle\phi_\mu|\hat{f}|\phi_\nu\rangle$), $\mathbb{S}$ je matice překryvových integrálů ($S_{\mu\nu} = \langle\phi_\mu|\phi_\nu\rangle$), $\mathbb{C}$ je matice LCAO rozvojových koeficientů (sloupce odpovídají jednotlivým MO) a ε je diagonální matice s energiemi molekulových orbitalů na diagonále.
Rovnice (9.3) představuje tzv. zobecněný vlastní problém. Budeme jej řešit pomocí funkce `eigh` matematické knihovny *Scipy*. Pro maticové elementy Fockova operátoru metody RHF v bázi AO platí [4]

$$F_{\mu\nu} = H_{\mu\nu} + \sum_{\lambda\sigma} P_{\lambda\sigma}\big[(\mu\nu|\sigma\lambda) - \frac{1}{2}(\mu\lambda|\sigma\nu)\big], \tag{9.4}$$

kde matice hustoty P pro systém s N elektrony je dána vztahem

$$P_{\lambda\sigma} = 2\sum_i^{N/2} C_{\lambda i}C^*_{\sigma i} \tag{9.5}$$

a pro dvouelektronové integrály v Mullikenově notaci platí

$$(\mu\nu|\sigma\lambda) = \int \phi^*_\mu(\mathbf{r}_1)\phi_\nu(\mathbf{r}_1)\frac{1}{r_{12}}\phi^*_\sigma(\mathbf{r}_2)\phi_\lambda(\mathbf{r}_2)d\mathbf{r}_1 d\mathbf{r}_2. \tag{9.6}$$

Pro celkovou elektronickou energii lze odvodit

$$E_{\text{el}} = \frac{1}{2}\sum_{\mu\nu} P_{\nu\mu}\big(H_{\mu\nu} + F_{\mu\nu}\big). \tag{9.7}$$

Selfkonzistentní procedura

Jelikož se jedná o nelineární rovnice (protože $\mathbb{F}(\mathbb{C})$), je nutné je řešit tzv. selfkonzistentní procedurou neboli SCF (*self-consistent field*), kterou je možné popsat následovně:

1. Počáteční odhad matice $\mathbb{F} = \mathbb{H}$.
2. Diagonalizace (zobecněný vlastní problém), zisk matice $\mathbb{C}$ a orbitálních energií ε.
3. Sestavení matice hustoty $\mathbb{P}$ podle (9.5).
4. Sestavení Fockovy matice $\mathbb{F}$ podle (9.4).
5. Kontrola konvergence $|E_{\text{new}} - E_{\text{old}}| < \text{Thresh}$:

   ANO Konec výpočtu.

   NE Zpět k bodu č. 2 a opakovat dokud nedojde ke konvergenci celkové energie.

9.2 Než začnete

1. Dosazením MO-LCAO rozvoje (9.2) do rovnice (9.1) odvoďte Roothaanovy rovnice (9.3).
2. Kolik neredundantních hodnot mají symetrické matice vzhledem k velikosti báze?
3. Zopakujte si permutační symetrii dvouelektronových integrálů v Mullikenově notaci (nezapomeňte, že pracujeme s reálnou bazí). Promyslete si, kolik jich bude a jak je budete ukládat v závislosti na tom jak budete sestavovat matici hustoty – můžete si zvolit jednu z dvou možností:
   - Přímočará implementace dle (9.4) (nápověda: dvouelektronové integrály uložíte do 2D matice).
   - Tzv. *integral driven algoritmus* kde máte uloženy hodnoty integrálů spolu s jejich indexy v listu, který při konstrukci $\mathbb{F}$ procházíte a na základě integrálních indexů přičítávate jejich hodnoty do elementů Fockovy matice.

   Který přístup je efektivnější a proč?

9.3 Struktura programu

Dostanete textový soubor INTDUMP pro molekulu HeH^+ (STO-3G), ve kterém je uložen počet elektronů, velikost báze, jaderná repulze a dále neredundantní jedno- a dvouelektronové integrály. Napište program, který tento soubor načte a spočte elektronickou energii metodou RHF.

Postup

1. Napište funkci, ktera rozparsuje vstupní soubor INTDUMP a načte všechnu potřebné hodnoty do proměnných a matic (viz podkapitola 2.10). Dbejte na to, aby tato funkce byla použitelná obecně, tj. pro libovolný počet elektronů a libovolně velkou bázi.
2. Proveďte kontrolní výpis a ověřte, že se matice načetly správně.
3. Napište funkce, které skonstrují matici hustoty a Fockovu matici.
4. Napište kód pro selfkonzistentní proceduru podle její popisu uvedeného výše. Ten se bude skládat z načítaní inputu a následně *while* smyčky, ve které budete kontrolovat, jestli energie skonvergovala.
5. V každé iteraci vypište celkovou energii $E_{\text{tot}} = E_{\text{el}} + E_{\text{nucl}}$ a rozdíl energií oproti předchozí iteraci.

Poznámky

- Konvergenční kritérium nastavte na Thresh $= 10^{-9}$ a použijte funkci `fabs` z knihovny `math` pro výpočet absolutní hodnoty.
- Pro řešení zobecněného vlastního problému použijte funkci `eigh` z knihovny `scipy.linalg`. Ta při volání `e,C = eigh(F,S)` vraci vektor `e` a matici `C`, kde v i-tém sloupci `C[:,i]` jsou uloženy LCAO koeficienty molekulových orbitalů prislušící vlastní hodnotě `e[i]`.
- Struktura vstupního souboru – dejte pozor:
  - Integrály jsou zde indexovány od 1, avšak Python indexuje pole od 0.
  - Soubor obsahuje všechny překryvové a jednoelektronové integrály, ale dvouelektronové integrály pouze pokud jsou větší než 10^{-7}.

```
ELECTRONS N_el BASIS N_bas          ! počet elektronů a bázových funkcí

OVERLAP                             ! překryvová matice ve tvaru S_μν = value_μν
  value_11
  value_12
  value_22
  ...
OVERLAP END

  value      μ    ν    σ    λ      ! dvouelektronové integrály ve tvaru (μν|σλ)
  ...
  value      μ    ν    0    0      ! jednoelektronové integrály ve tvaru H_μν
  ...
  value      0    0    0    0      ! jaderná repulze E_nucl
```

- Kopii matice `A` vytvoříte jako `B = A.copy()`
- Pro účely testování si přidejte možnost nastavit maximální počet iterací. Dále jsou zde uvedeny energie E_{tot} a matice $\mathbb{F}$ a $\mathbb{P}$ z prvních tří iterací.

```
   ── Iter 1 ──          ── Iter 2 ──          ── Iter 3 ──
E(tot) = -2.774773    E(tot) = -2.859428    E(tot) = -2.860458

F: -2.6527 -1.3472    F: -1.3903 -0.9731    F: -1.4513 -1.0434
   -1.3472 -1.7318       -0.9731 -0.7429       -1.0434 -0.8034

P:  1.7267  0.2598    P:  1.3342  0.5166    P:  1.2899  0.5384
    0.2598  0.0391        0.5166  0.2000        0.5384  0.2247
```

- Při implementaci některých vztahů můžete pro elegantnější zápis využít maticový součin (`A@B`) a stopu matice (`numpy.trace(A)`) místo sumace pomocí for cyklů.

Literatura

[1] D. Martínek *Přehled základních Linuxových příkazů* (2004):
`http://www.fit.vutbr.cz/~martinek/linux/files/linuxref.pdf`

[2] Guido van Rossum and the Python development team *Python Tutorial – Release 3.5.6*:
`https://docs.python.org/3.5/tutorial/`

[3] *Klíčová slova programu Gaussian*:
`http://www.gaussian.com/g_tech/g_ur/l_keywords09.htm`

[4] A. Szabo and N. Ostlund *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory* Dover, New York (1996).

[5] J. Keeler *Understanding NMR Spectroscopy* (2002):
`http://www-keeler.ch.cam.ac.uk/lectures/Irvine/`

[6] Fotochemická izomerizace benzenu:
`http://www.ch.ic.ac.uk/robb/casscf_benzene_handout.php`

[7] Výpočty UV spekter:
`http://www.chem.ucsb.edu/~kalju/chem226/public/task2C.html`