Universitatea Babeș-Bolyai Cluj-Napoca

Facultatea de Fizică

Specializarea Fizică Informatică

LUCRARE DE LICENȚĂ

Coordonator științific

Prof. dr. Vasile CHIȘ

Absolvent

Patrick CĂRUNTU

Cluj-Napoca

2025

Universitatea Babeș-Bolyai Cluj-Napoca

Facultatea de Fizică

Specializarea Fizică Informatică

LUCRARE DE LICENȚĂ

Aplicație informatică pentru automatizarea calculului populațiilor Boltzmann ale conformerilor și tautomerilor moleculari

Coordonator științific

Prof. dr. Vasile CHIȘ

Absolvent

Patrick CĂRUNTU

Cluj-Napoca

2025

Cuprins

Abstract

This thesis presents the design and implementation of a software solution that automates the calculation of Boltzmann populations for molecular systems. The aim of this project is to simplify the computational workflow used in molecular spectroscopy and quantum chemistry, by developing a user-friendly application that can process Gaussian output files and compute the populations of energy levels based on temperature.

The application combines principles from molecular physics and computational chemistry with modern programming practices in Python. Using output data from quantum chemistry simulations (performed using Gaussian), the software calculates the Boltzmann distribution over electronic or vibrational states of molecules. It enables researchers to visualize the distribution graphically and to analyze population ratios in thermally excited systems.

The system is structured modularly with a clean interface, separating back-end logic, file handling, and user interface, and was developed using libraries such as matplotlib, numpy, and customtkinter. The software was validated through case studies involving known molecules and showed excellent correlation with theoretical expectations.

This automation represents a valuable tool for both educational and research purposes, providing accurate and fast calculations in a field where manual computation would be inefficient or prone to error.

Rezumat

Această lucrare de licență prezintă proiectarea și implementarea unei aplicații software pentru automatizarea calculului populațiilor Boltzmann pentru sisteme moleculare. Scopul proiectului este de a simplifica fluxul de lucru din spectroscopia moleculară și chimia cuantică, prin dezvoltarea unei aplicații intuitive care poate procesa fișierele de ieșire din Gaussian și calcula populațiile Boltzmann ale conformerilor moleculari în funcție de temperatură.

Aplicația îmbină concepte din fizica moleculară și chimia computațională cu practici moderne de programare în Python. Folosind datele obținute din simulări cuantice (Gaussian), software-ul calculează distribuția Boltzmann pe stările electronice sau vibraționale ale moleculelor. Utilizatorul poate vizualiza grafic distribuția și analiza rapoartele populaționale în funcție de temperatura aleasă.

Sistemul este dezvoltat modular, cu o separare clară între logică, interfața grafică și manipularea fișierelor, utilizând biblioteci precum matplotlib, numpy și customtkinter. Aplicația a fost validată prin studii de caz pe molecule cunoscute, demonstrând o concordanță excelentă cu așteptările teoretice.

Această automatizare se dovedește un instrument valoros atât pentru scopuri educaționale, cât și pentru cercetare, oferind calcule rapide și precise într-un domeniu unde metodele manuale sunt ineficiente sau predispuse la erori.

Introducere

Distribuția Boltzmann joacă un rol fundamental în înțelegerea comportamentului sistemelor moleculare aflate în echilibru termodinamic. Aceasta oferă o metodă elegantă de a determina modul în care populațiile conformerilor moleculare sunt distribuite între diferitele stări de energie, în funcție de temperatură. Această relație are implicații directe în spectroscopie, chimie cuantică, fizică moleculară și chiar biologie structurală [Atk21].

În mod tradițional, calculul populațiilor Boltzmann presupune accesul la energii relative ale stărilor moleculare, urmat de aplicarea formulelor statistice. De cele mai multe ori, aceste energii sunt obținute din simulări cuantice complexe efectuate cu ajutorul unor programe precum Gaussian [Gau16]. Prelucrarea manuală a fișierelor de ieșire din aceste simulări este însă un proces laborios și predispus la erori.

În acest context, automatizarea calculelor Boltzmann devine esențială, mai ales în laboratoare unde se analizează un număr mare de molecule sau se testează variația populațiilor în funcție de temperatură. Prin intermediul unei aplicații software dedicate, acest proces poate fi simplificat considerabil, oferind utilizatorului atât viteză de analiză cât și o interfață grafică intuitivă.

Lucrarea de față își propune dezvoltarea și validarea unei aplicații Python care permite extragerea automată a nivelurilor de energie din fișierele de ieșire Gaussian și calculul populațiilor Boltzmann corespunzătoare. Proiectul aduce împreună concepte teoretice din fizica moleculară și abordări moderne de programare, rezultând într-un instrument didactic și de cercetare valoros.

Structura lucrării reflectă parcursul dezvoltării aplicației: de la fundamentele teoretice ale distribuției Boltzmann și detalii computaționale legate de metodele de simulare, până la implementarea practică a soluției software și analiza unor studii de caz relevante.

Populațiile Boltzmann sunt extrem de utile pentru interpretarea proprietăților spectroscopice atunci când în proba analizată există un amestec de conformeri și/sau tautomeri moleculari. Răspunsul spectroscopic este afectat de neomogenitatea probei analizate și interpretarea corectă a acestuia depinde de proprietățile componentelor individuale și de contribuția fiecărei componente [Li19, Yan08].

1. Calculul populatiei Boltzmann

1.1 Istoric și fundamente

Distribuția Boltzmann a fost formulată în secolul al XIX-lea de Ludwig Boltzmann, în cadrul dezvoltării mecanicii statistice. Ea oferă o punte între nivelul microscopic — descris prin stările cuantice ale particulelor [Atk21] — și proprietățile macroscopice observabile ale sistemelor, precum presiunea sau temperatura.

Conceptul central este acela că la o anumită temperatură T, moleculele unui sistem nu ocupă doar starea fundamentală, ci sunt distribuite între toate stările accesibile, cu o probabilitate care scade exponențial cu energia stării. Această distribuție reflectă echilibrul dintre tendința naturală a moleculelor de a ocupa stări joase de energie și agitația termică.

1.2 Fundamente teoretice ale distributie Boltzmann

Distribuția Boltzmann este una dintre cele mai importante relații în termodinamică și mecanică statistică, descriind probabilitatea de ocupare a unei stări energetice de către o particulă (sau un sistem) aflat în echilibru termic. În contextul fizicii moleculare, această distribuție explică modul în care moleculele sunt împărțite între stări fundamentale și stări excitate în funcție de temperatură.

Formula distribuției Boltzmann este dată de [McQ00]:

$$P_i = \frac{e^{-E_i/k_B T}}{Z}$$

unde:

- P_i este probabilitatea ca o moleculă să se afle în starea i,
- E_i este energia stării i,
- k_B este constanta lui Boltzmann,
- T este temperatura absolută,

- Z este funcția de particiție (partition function), care asigură normalizarea:

$$Z = \sum_{i=1}^{n} e^{-E_i/k_B T}$$

Funcția de particiție are un rol esențial în determinarea proprietăților termodinamice macroscopice ale unui sistem, cum ar fi energia internă, entropia și capacitatea termică.

1.3 Aplicații ale distribuției Boltzmann

Distribuția Boltzmann este esențială în domenii precum:
- Spectroscopia vibrațională și electronică: determinarea intensității relative a benzilor de absorbție sau emisie;
- Cinética chimică: evaluarea fracțiunii de molecule cu energie suficientă pentru a reacționa;
- Fizica plasmei și astrofizică: calculul populațiilor nivelurilor energetice în stele sau nebuloase;
- Biofizică: studiul populației conformațiilor moleculare în funcție de temperatura mediului.

1.4 Limitări și ipoteze

Distribuția Boltzmann se aplică în sistemele:
- aflate în echilibru termic;
- rarefiate (interacțiuni slabe între particule);
- unde se poate considera că energia fiecărei stări este bine definită și distinctă.

Ea nu se aplică direct în sistemele cu interacțiuni puternice sau în afara echilibrului, cazuri în care trebuie folosite alte distribuții statistice (ex. distribuția Fermi-Dirac sau Bose-Einstein).

În contextul acestei lucrări, distribuția Boltzmann este utilizată pentru a calcula populația

relativă a stărilor energetice obținute din simulări cuantice [Lev05]. Aceste populații sunt apoi reprezentate grafic și utilizate în analiza comportamentului termic al moleculelor.

1.5 Exemple aplicate

Un exemplu comun de aplicare este în spectroscopia vibrațională: la temperaturi joase, majoritatea moleculelor se găsesc în starea vibrațională fundamentală (v = 0), dar la temperaturi ridicate, devin populate și stările excitate (v = 1, v = 2 etc.), influențând forma și intensitatea benzilor spectrale.

Un alt exemplu este distribuția populațiilor pe nivelele electronice ale ionilor metalici în stele, de unde se pot deduce temperaturile plasmei pe baza intensităților liniilor spectrale (legea Boltzmann aplicată în astrofizică).

1.7 Interpretare statistică

Distribuția Boltzmann reflectă probabilitatea relativă ca o moleculă aleatoare să ocupe o stare anume. Ea nu determină poziția sau energia exactă a unei molecule, ci doar tendința sistemului în ansamblu. Această abordare statistică devine esențială când lucrăm cu un număr foarte mare de particule, tipic în chimie și fizică moleculară.

De asemenea, contribuția fiecărei stări la proprietățile termodinamice depinde nu doar de energia acesteia, ci și de degenerescență. Stările cu energii mai ridicate, dar cu degenerescență mare, pot avea o populație semnificativă.

1.8 Relevanța în automatizare

Automatizarea calculului distribuției Boltzmann presupune extragerea valorilor E_i și g_i din date cuantice (ex. fișiere Gaussian), aplicarea formulei, și obținerea populațiilor pentru o gamă de temperaturi. Acest lucru este important deoarece în mod obișnuit sunt implicate zeci sau sute de stări, iar calculul manual devine imposibil de susținut.

Aplicația propusă în această lucrare gestionează aceste etape în mod automat, oferind o

interfață prietenoasă utilizatorului și rezultate prezentate sub formă grafică. Ea permite analiza imediată a efectului temperaturii asupra distribuției populaționale, fiind utilă atât în cercetare, cât și în educație.

2. Detalii computaționale și descrierea aplicației

În acest capitol sunt descrise detaliile computaționale și aspectele tehnice ale aplicatiei dezvoltate pentru automatizarea calculului populației Boltzmann. Aplicația a fost concepută pentru a procesa fișierele de ieșire obținute în urma simulărilor cu Gaussian, din care se extrage automat nivelurile energetice pentru calcul si structura moleculară pentru a o inspecta.

2.1: Structura aplicației

Aplicația este dezvoltata in limbajul Python [PytDoc] și organizata în module separate:

- App/ : conține interfața grafică utilizator realizată cu ajutorul bibliotecii customtkinter;
- Backend/ : conține logica de calcul și de procesare a fișierilor de ieșire Gaussian;
- main.py : inițializează aplicația
- mac.command: pornește aplicația pentru calculatoarele cu sistem de operare MacOS;
- windows.bat: pornește aplicația pentru calculatoarele cu sistem de operare Windows

Această structura oferă claritate și ușurință în întreținere și extindere.

2.2: Tehnologii utilizate

Aplicația utilizează urmatoarele biblioteci:

- numpy: pentru calcule matematice [Nump]
- matplotlib: pentru generarea graficelor si inspectarea moleculei [Matp]
- customtkinter: pentru generarea interfaței grafice utilizator [CustTk]
- re: pentru procesarea textului și extragerea datelor relevante din fișierele .log
- os: pentru extragerea conținutului directivelor

Aplicația este portabilă și poate fi rulată pe orice sistem care are instalat Python 3.10+.

2.3: Interacțiunea cu utilizatorul

Interfața grafică este concepută intuitiv cu ecrane dedicate pentru:

- selectarea fișierelor de input
- configurarea listei de fișiere de input
- vizualizarea structurii moleculei
- analiza nivelurilor energetice
- afișarea rezultatelor calculului populațiilor Boltzmann

Utilizatorul poate comuta rapid între diferite module, fără a necesita cunoștințe de programare sau operare complexă.

2.4: Considerații privind performanța

Algoritmii sunt optimizați pentru procesarea rapidă a fișierelor de mari dimensiuni, folosind expresii regulate și parsare secvențială. De asemenea, aplicația gestionează erorile de citire sau datele incomplete prin afișarea mesajelor în terminal fără a închide aplicația.

Performanța a fost validată folosind fișierele de ieșire Gaussian conținând zeci de mii de linii de informație, fără întârzieri perceptibile în timpul execuției.

3. Implementarea și funcționalitățile aplicației

Acest capitol detaliază implementarea efectivă a aplicației pentru automatizarea calculului populației Boltzmann, prezentând organizarea codului, clasele principale și fluxul logic de execuție.

3.1: Interpretarea energiilor din fișierele Gaussian

În fișierile .log, datele referitoare la energii se află în urmatoarele secvențe:

```
Zero-point correction=                           0.183820 (Hartree/Particle)
Thermal correction to Energy=                    0.194828
Thermal correction to Enthalpy=                  0.195772
Thermal correction to Gibbs Free Energy=         0.146187
Sum of electronic and zero-point Energies=           -571.343480
Sum of electronic and thermal Energies=              -571.332472
Sum of electronic and thermal Enthalpies=            -571.331527
Sum of electronic and thermal Free Energies=         -571.381112
```

și

```
Unable to Open any file for archive entry.
1\1\GINC-COMPUTE075\SP\RAPFD TD-FC\6-311++G(2df,p)\C10H10N2O1\VCHIS\06
-Jul-2024\0\\#p td(nstates=30) geom=check apfd/6-311++G(2df,p) scf=ver
ytight int=(grid=superfine, acc2e=12) scrf=(solvent=water,smd)\\edavar
one_c1 amine OFRTD\\0,1\C,0,-1.1847865072,1.1957265815,-0.4304144812\C
,0,-2.5448601694,0.7519637853,-0.4654279843\C,0,-2.577310387,-0.546259
7744,-0.0511289965\N,0,-1.3087239724,-0.9721163094,0.1768443607\N,0,-0
.4641110805,0.1063790365,0.0526683278\O,0,-0.6678193023,2.274143877,-0
.7719674149\C,0,-3.7214218311,-1.4574892338,0.1677413157\C,0,0.9266546
946,-0.0811961261,0.0558607829\C,0,1.469902379,-1.3143852193,-0.298481
3517\C,0,2.8462904377,-1.4879970979,-0.2743796475\C,0,3.6840056534,-0.
4399177294,0.0858501915\C,0,3.1327047868,0.7864375554,0.4384688739\C,0
,1.7580120612,0.9703759002,0.4366489194\H,0,-3.382316909,1.3491705101,
-0.7898567449\H,0,-3.7712995119,-1.7702276458,1.2149200331\H,0,-3.6231
663164,-2.3595012973,-0.4420534584\H,0,-4.6540237661,-0.9576638239,-0.
09112899\H,0,0.8223956673,-2.130291136,-0.5986129832\H,0,3.264249416,-
2.4508194834,-0.5497661892\H,0,4.7597727288,-0.5794519518,0.0974397453
\H,0,3.7767962777,1.6084727234,0.7334940109\H,0,1.3275348407,1.9160952
964,0.7385315679\H,0,-1.0807241699,-1.6423144675,0.9029807927\\Version
=ES64L-G16RevC.01\State=1-A\HF=-571.5272995\RMSD=4.905e-09\PG=C01 [X(C
10H10N2O1)]\\@
The archive entry for this job was punched.
```

Pentru a căuta energiile se folosesc algoritmii aflate în Backend/Loader/FileLoader.py:

```
def load_properties(f):
    l = 0
    energies = []
    with open(f, 'r') as file:
        for line in file:
            if line[:23] == " Zero-point correction=":
                l = 1
            if l == 1:
                if line == ' \n':
                    break
                else:
                    energy = float(line.split('=')[1].strip().split('
')[0])
                    energies.append(energy)
        return energies
```

Explicarea funcției:

Pasul 1:

Parcurge fiecare linie până ajunge la linia care definește Zero-point correction=, ce pornește algoritmul de salvare a valorilor energetice.

Pasul 2:

Verifică dacă linia citită nu conține informație, ce înseamnă că a parcurs toate liniile ce conțin informație despre energii, astfel returnând toate datele citite.

Pasul 3:

Dacă linia conține informație, împarte linia citită în două valori despărțită de =.

Pasul 4:

Reține a doua valoare (valoarea energiei) în memorie, trecând la următoarea linie și repetând pașii 2, 3, 4.

Pentru găsirea șirului "HF" trebuie de analizat mai multe linii deoarece valoarea energiei poate fi redată în 2 rânduri.

```
def load_hf(f):
    with open(f, 'r') as file:
        checker = 0
        data = ""
        for line in file:
            line = line.strip().strip('\n')
            if line == "The archive entry for this job was punched.":
                checker = 0
                verify, result = find_hf(data)
                if verify:
                    return result
                else:
                    data = ""
            if checker == 1:
                data += line
            if line == "Unable to Open any file for archive entry.":
                checker = 1
```

```
def find_hf(d):
   st = d.split('\\')
   for data in st:
       if data[:3] == "HF=":
           return True, data.split("=")[1]
   return False, None
```

Explicarea funcției:

Pasul 1:

Trece prin fiecare linie a fișierului de ieșire până ajunge la linia “Unable to Open any file for archive entry”, ce schimba valoarea lui checker astfel înțelegând că a ajuns la liniile unde se ascunde informația despre HF.

Pasul 2:

Iterează fiecare linie până la linia de “The archive entry for this job was punched”, reținând toate liniile precedente.

Pasul 3:

Împarte aceste linii în secvențe de informații despărțite de “\\”.

Pasul 4:

Verifica dacă în una din aceste secvențe există informația despre HF. Dacă a găsit valoarea lui HF, returneaza valoarea respectivă și o valoare Bool care confirmă găsirea valoarii și oprește funcția. Dacă nu a găsit valoare lui HF, returnează un Bool care indică faptul că informația despre HF nu este prezentă în aceste linii și continuă căutarea de la Pasul 1.

3.2: Calculul populației Boltzmann

După cum am discutat în Capitolul 1, formula folosită pentru a calcula populația Boltzmann este:

$$P_i = \frac{e^{-E_i/k_B T}}{Z}$$

Din cauza că energiile sunt negative și de ordinul 10^2, toți factorii $e^{-E_i/k_B T}$ vor fi prea mari pentru a-i putea calcula. Astfel, înmulțim ambele părți ale fracției cu factorul Boltzmann a celei mai mici energii dintre cele extrase, obținând:

$$P_i = P_i \frac{e^{-E_{min}/k_B T}}{e^{-E_{min}/k_B T}} = P_i \frac{e^{E_{min}/k_B T}}{e^{E_{min}/k_B T}} = \frac{e^{-(E_i - E_{min})/k_B T}}{\sum_{j=1}^{n} e^{-(E_j - E_{min})/k_B T}}$$

Noul factor Boltzmann căutat va fi: $e^{-(E_i - E_{min})/k_B T}$

Metoda de calcul folosită in aplicație se află în Backend/Functions/Function.py:

```
def population_calc(energies, temperature):

    kb = 0.0019872041

    T = temperature

    c = 627.51

    best = min(energies)

    deltas = [(x - best) * c for x in energies]

    boltzmann_factors = [exp(0 - x / (kb * T)) for x in deltas]

    population = [round(x * 100 / sum(boltzmann_factors), 3) for x in
boltzmann_factors]

    return deltas, boltzmann_factors, population
```

Acest algoritm permite calculul populației Boltzmann în dependență de temperatură:

Pasul 1:

Determină cea mai mică energie din cele calculate

Pasul 2:

Calculează ΔE_i pentru fiecare conformer

Pasul 3:

Calculează factorul Boltzmann pentru fiecare conformer

Pasul 4:

Calculează distribuția Boltmann pentru fiecare conformer după care returnează toate datele obținute

3.3: Interpretarea datelor structurii moleculare din fișierele Gaussian

Pentru extragerea structurii moleculare, este necesar de căutat poziția fiecărui atom precum și conexiunile acestuia. Primul pas este de a analiza din ce atomi e formată molecula, ceea ce putem găsi în următoarele linii:

```
Structure from the checkpoint file:  "/sandata1/users-data/vchis/balan/eda_c1_amine_w_OFRTD.chk"
Charge =  0 Multiplicity = 1
Redundant internal coordinates found in file.  (old form).
C,0,-1.1847865072,1.1957265815,-0.4304144812
C,0,-2.5448601694,0.7519637853,-0.4654279843
C,0,-2.577310387,-0.5462597744,-0.0511289965
N,0,-1.3087239724,-0.9721163094,0.1768443607
N,0,-0.4641110805,0.1063790365,0.0526683278
O,0,-0.6678193023,2.274143877,-0.7719674149
C,0,-3.7214218311,-1.4574892338,0.1677413157
C,0,0.9266546946,-0.0811961261,0.0558607829
C,0,1.469902379,-1.3143852193,-0.2984813517
C,0,2.8462904377,-1.4879970979,-0.2743796475
C,0,3.6840056534,-0.4399177294,0.0858501915
C,0,3.1327047868,0.7864375554,0.4384688739
C,0,1.7580120612,0.9703759002,0.4366489194
H,0,-3.382316909,1.3491705101,-0.7898567449
H,0,-3.7712995119,-1.7702276458,1.2149200331
H,0,-3.6231663164,-2.3595012973,-0.4420534584
H,0,-4.6540237661,-0.9576638239,-0.09112899
H,0,0.8223956673,-2.130291136,-0.5986129832
H,0,3.264249416,-2.4508194834,-0.5497661892
H,0,4.7597727288,-0.5794519518,0.0974397453
H,0,3.7767962777,1.6084727234,0.7334940109
H,0,1.3275348407,1.9160952964,0.7385315679
H,0,-1.0807241699,-1.6423144675,0.9029807927
Recover connectivity data from disk.
```

Primul șir de caractere divizat de “,” reprezintă atomul, iar numărul liniei pe care este scris reprezintă numărul de ordine al acestuia (Center Number înregistrat de Gaussian (ex.: 1-C, 2-C, 3-C, 4-N, …, 23-H). Numărul de ordine va fi necesar de extras pentru extragerea informațiilor adiționale a atomilor. Șirurile de caractere de pe a treia poziție și mai departe reprezintă pozițiile atomilor în coordonate carteziene, însă în acest caz ele sunt puțin schimbate pentru alte calcule. Din această secvență avem nevoie să extragem doar atomii și numărul acestora de ordine. Pentru a găsi secvența se caută linia “Redundant internal coordinates found in file. (old form).” și se înregistrează informația pâna la linia “Recover connectivity data from disk.”

Al doilea pas este de a găsi pozițiile acestor atomi, în coordonate carteziene, în orientarea standard a moleculei.

Standard orientation:

Center Number	Atomic Number	Atomic Type	Coordinates (Angstroms) X	Y	Z
1	6	0	-1.197621	1.295617	-0.211442
2	6	0	-2.566965	0.880569	-0.235636
3	6	0	-2.611414	-0.449423	0.059707
4	7	0	-1.344563	-0.916310	0.200206
5	7	0	-0.482807	0.154232	0.143100
6	8	0	-0.671012	2.391695	-0.472581
7	6	0	-3.765564	-1.356924	0.237511
8	6	0	0.903005	-0.056915	0.077013
9	6	0	1.407599	-1.261635	-0.408237
10	6	0	2.780083	-1.460548	-0.451706
11	6	0	3.651652	-0.464598	-0.029269
12	6	0	3.138808	0.733615	0.453945
13	6	0	1.769030	0.940737	0.520464
14	1	0	-3.402963	1.519435	-0.472705
15	1	0	-3.784393	-1.763476	1.253010
16	1	0	-3.707872	-2.200773	-0.455107
17	1	0	-4.696294	-0.819379	0.060389
18	1	0	0.733029	-2.035232	-0.756873
19	1	0	3.167914	-2.401152	-0.829073
20	1	0	4.724028	-0.623226	-0.070806
21	1	0	3.809911	1.513843	0.798186
22	1	0	1.369286	1.862099	0.923112
23	1	0	-1.104635	-1.654131	0.853259

Din această secvență putem extrage coordonatele x, y și z a atomilor prin anumite manipulări a șirului de caractere. Aici putem observa necesitatea înregistrării numărului de ordine a atomilor. Pentru a găsi acest tabel se caută linia “Standard Orientation” iar după 4 linii se înregistrează informația pană la sfârșitul tabelului

Ultimul pas necesar pentru extragerea completă a informației despre structura moleculară este de a găsi legăturile dintre atomi.

```
MicOpt= -1 -1 -1
                                  ----------------------------
                                  !    Current Parameters    !
                                  ! (Angstroms and Degrees)  !
--------------------------                                    --------------------------
! Name  Definition              Value                                                  !
-------------------------------------------------------------------------------------
! R1    R(1,2)                  1.4311                                                 !
! R2    R(1,5)                  1.3926                                                 !
! R3    R(1,6)                  1.2437                                                 !
! R4    R(2,3)                  1.3631                                                 !
! R5    R(2,14)                 1.0785                                                 !
! R6    R(3,4)                  1.3574                                                 !
! R7    R(3,7)                  1.4789                                                 !
! R8    R(4,5)                  1.3755                                                 !
! R9    R(4,23)                 1.0141                                                 !
! R10   R(5,8)                  1.4034                                                 !
! R11   R(7,15)                 1.094                                                  !
! R12   R(7,16)                 1.0932                                                 !
! R13   R(7,17)                 1.0893                                                 !
! R14   R(8,9)                  1.3934                                                 !
! R15   R(8,13)                 1.3935                                                 !
! R16   R(9,10)                 1.3875                                                 !
! R17   R(9,18)                 1.084                                                  !
! R18   R(10,11)                1.3892                                                 !
! R19   R(10,19)                1.0852                                                 !
! R20   R(11,12)                1.39                                                   !
! R21   R(11,20)                1.0848                                                 !
! R22   R(12,13)                1.3869                                                 !
! R23   R(12,21)                1.0852                                                 !
! R24   R(13,22)                1.082                                                  !
! A1    A(2,1,5)                104.9364                                               !
```

Acest tabel reprezintă parametrii geometrici ai moleculei analizate, unde coloana “Name” reprezintă numele parametrului studiat (R1, A1, D1), coloana Definition reprezintă:

- R(i, j) reprezintă lungimea legăturii dintre atomii i și j
- A(i, j, k) reprezintă unghiul format de atomii i, j și k
- D(i, j, k, l) reprezintă unghiul diedru determinat de patru atomi i, j, k și l

iar coloana “Value” reprezintă valoarea acestora.

Pentru a extrage legăturile între atomi avem nevoie de parantezele din coloana “Definition” cu parametrul R. Tabelul îl găsim prin căutarea liniilor ce încep cu caracterul “!” și verificam ca următorul caracter (cu excepția spațiului) să fie R.

```
def load_molecule(f):
    with open(f, 'r') as file:
        i, j, k, l = 0, 1, 0, 0
        atoms = {}
        bonds = []
        for line in file:
            if line == " Recover connectivity data from disk.\n" and
i == 2:
                i = 0
                k = 1
            if i == 2:
                atom = line.strip(" ").strip("\n").split(",")
                atoms[j] = [atom[0]]
                j += 1

            if line == " Redundant internal coordinates found in
file.  (old form).\n":
                i += 1

            if l == 1:
```

```
                line = line.strip(' ').strip("\n").strip("\t")
                try:
                    if line[2] == "A":
                        break
                    if line[2] == "R":
                        bond = []
                        for index in range(len(line)):
                            if line[index] == "(":
                                left = index + 1
                            if line[index] == ")":
                                right = index
                                bond_char =
line[left:right].split(',')
                                for x in bond_char:
                                    bond.append(int(x))
                                bonds.append(bond)
                                break
                except:
                    pass
            if k == 1:
                try:
                    if line[1] == "!":
                        l = 1
                except:
                    pass
```

```
        i, j = 0, 0
        for line in file:
            line = line.strip('\n').strip()
            if line == "Standard orientation:":
                i = 1
            if j == 5 and line ==
"---------------------------------------------------------------------
--":
                break
            elif j == 5:
                line = line.split(' ')
                num = int(line[0])
                for k in range(3):
                    line = ' '.join(line[1:]).strip().split(' ')
                x = float(line[0])
                line = ' '.join(line[1:]).strip().split(' ')
                y = float(line[0])
                line = ' '.join(line[1:]).strip().split(' ')
                z = float(line[0])
                atoms[num].append([x, y, z])
            elif i == 1:
                j += 1


        return atoms, bonds
```

3.4: Rularea aplicației

Pentru a deschide aplicația, este necesară descărcarea librăriile externe utile. Acest pas este automatizat în fișierele de rulare în funcție de Sistemul de Operare folosit.

Pentru Windows se folosește un fișier de tip .bat:

```
@echo off
SETLOCAL

SET SCRIPT_DIR=%~dp0
SET MAIN_PY=%SCRIPT_DIR%TezaOficial\main.py

echo Script is running from: %SCRIPT_DIR%
python --version

echo Installing required libraries...
python -m pip install --upgrade pip
python -m pip install customtkinter tkinterdnd2 matplotlib numpy

IF EXIST "%MAIN_PY%" (
   echo Running main.py...
   python "%MAIN_PY%"
) ELSE (
   echo Could not find main.py at %MAIN_PY%
   pause
   EXIT /B 1
```

```
)


ENDLOCAL

pause
```

Pentru Mac se folosește un fișier de tip .command:

```
#!/bin/bash


# Resolve the absolute path of the script

SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"


echo "Script is running from: $SCRIPT_DIR"


# Check if Python 3 is installed

if ! command -v python3 &>/dev/null; then

    echo "Python3 not found. Installing..."

    sudo apt update

    sudo apt install -y python3 python3-pip

else

    echo "Python3 found: $(python3 --version)"

fi


echo "Installing required Python packages..."

pip3 install --upgrade pip

pip3 install customtkinter tkinterdnd2 matplotlib numpy
```

```
# Run the main Python script
MAIN_PY="$SCRIPT_DIR/main.py"
if [ -f "$MAIN_PY" ]; then
    echo "Running main.py..."
    python3 "$MAIN_PY"
else
    echo "Error: Could not find main.py in $MAIN_PY"
    exit 1
fi
```

Pentru Linux se folosește un fișier de tip .sh:

```
#!/bin/bash
SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"

echo "Script is running from: $SCRIPT_DIR"

# Check if Python 3 is installed
if ! command -v python3 &>/dev/null; then
    echo "Python3 not found. Installing..."
    sudo apt update && sudo apt install -y python3 python3-pip
else
    echo "Python3 found: $(python3 --version)"
fi
```

```
echo "Installing required Python packages..."

pip3 install --upgrade pip

pip3 install customtkinter tkinterdnd2 matplotlib numpy


MAIN_PY="$SCRIPT_DIR/TezaOficial/main.py"

if [ -f "$MAIN_PY" ]; then

   echo "Running main.py..."

   python3 "$MAIN_PY"

else

   echo "Error: Could not find main.py in $MAIN_PY"

   exit 1

fi
```

Dacă se folosește Linux, pentru a putea rula aplicația cu ajutorul unui double-click, trebuie de deschis terminalul și apoi rulată comanda:

```
chmod +x *directory_path*/LINUX.sh
```

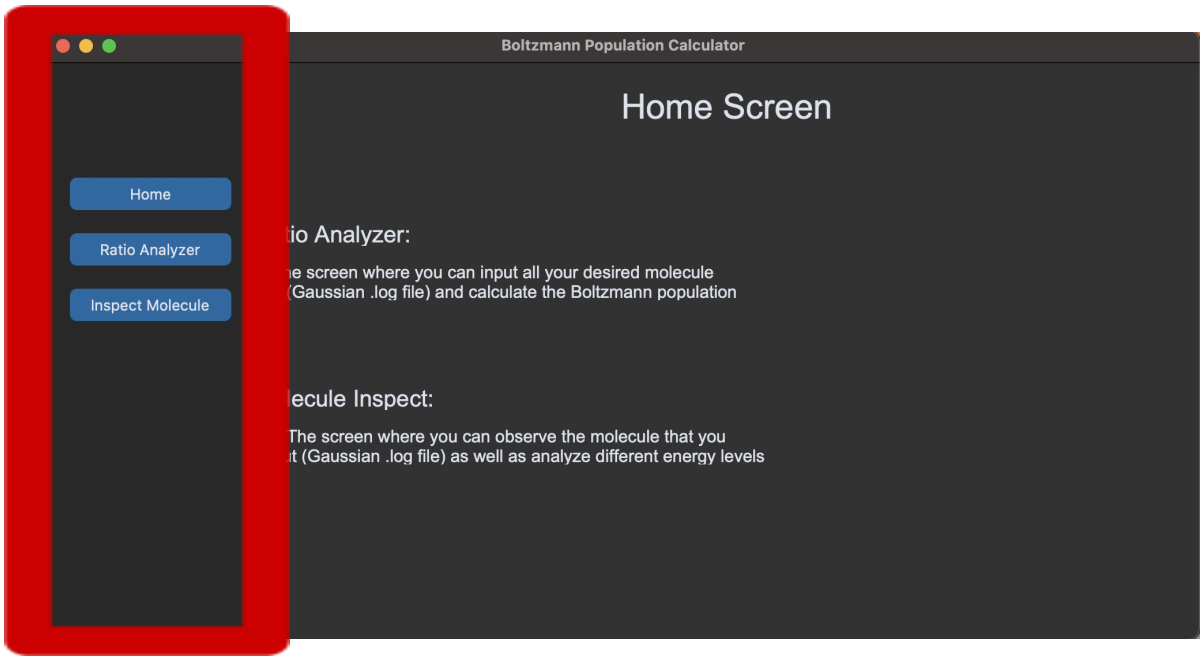
4. Interfața Grafică a Aplicației

În acest capitol se vor descrie metodele folosite pentru afișarea informației în aplicație a fiecărei pagini.

4.1 Ecranul Principal

Ecranul principal se împarte în două secțiuni:

- Ecranul de navigare
- Ecranul de afișare a informației



Ecranul de navigare reprezintă interfața care permite utilizatorului să acceseze diverse secțiuni sau funcționalități ale aplicației. În cazul de față, acest ecran include butoanele “Home”, “Ratio Analyzer” și “Inspect Molecule”, care permit redirecționarea spre alte ecrane ale aplicației. Ecranul de navigare rămâne mereu constant, butoanele schimbând doar ecranul de afișare a informației.

Importuri folosite:

```
from App.Screens.MainScreen import MainScreen

from App.Screens.CalculatorScreen import CalculatorScreen

from App.Screens.MoleculeScreen import MoleculeScreen

from tkinterdnd2 import TkinterDnD

import customtkinter as ctk
```

Pentru a ascunde toate ecranele de afișare se folosește:

```
def hide_all_frames(self):

   for frame in (self.home_frame, self.calculator_frame,
self.molecule_frame):

       frame.pack_forget()
```

Pentru a afișa ecranul “Home”:

```
def show_home(self):

   self.hide_all_frames()

   self.home_frame.pack(fill="both", expand=True)
```

Pentru a afișa ecranul “Ratio Analyzer”:

```
def show_dnd(self):

   self.hide_all_frames()

   self.calculator_frame.pack(fill="both", expand=True)
```

Pentru a afișa ecranul “Inspect Molecule”:

```
def show_graph(self):

   self.hide_all_frames()
```

```
    self.molecule_frame.pack(fill="both", expand=True)
```

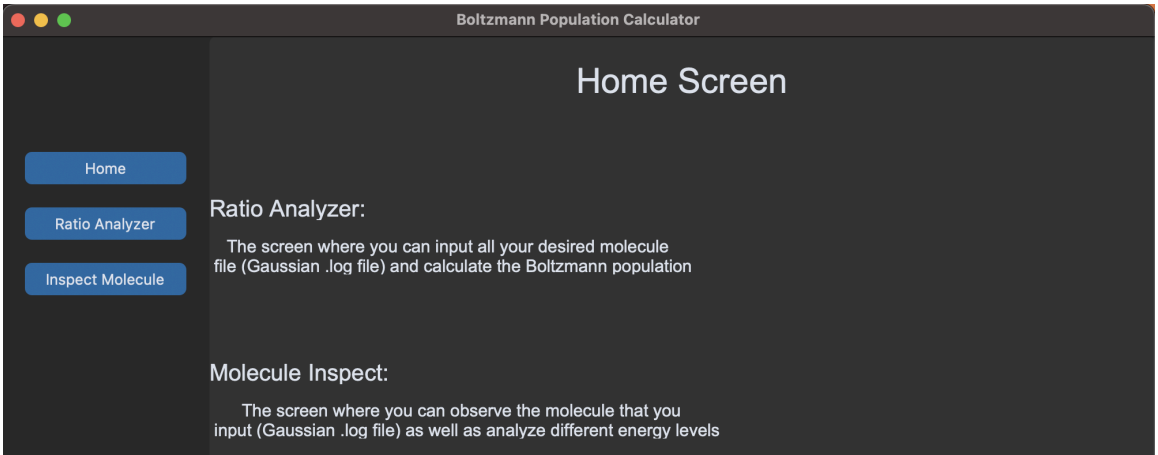
Pentru a inițializa ecranele și de a configura butoanele:

```
class App(TkinterDnD.Tk):

    def __init__(self):

        super().__init__()

        self.title("Boltzmann Population Calculator")

        self.geometry("1000x500")


        # Configure the grid layout

        self.grid_rowconfigure(0, weight=1)

        self.grid_columnconfigure(1, weight=1)  # main content area


        # Sidebar Frame

        self.sidebar = ctk.CTkFrame(self, width=200, corner_radius=0)

        self.sidebar.grid(row=0, column=0, sticky="nswe")

        self.sidebar.grid_rowconfigure(5, weight=1)


        # Sidebar buttons

        self.home_btn = ctk.CTkButton(self.sidebar, text="Home",
command=self.show_home)

        self.home_btn.grid(row=0, column=0, padx=20, pady=(100, 10))


        self.dnd_btn = ctk.CTkButton(self.sidebar, text="Ratio
Analyzer", command=self.show_dnd)
```

```
        self.dnd_btn.grid(row=1, column=0, padx=20, pady=10)


        self.mol_btn = ctk.CTkButton(self.sidebar, text="Inspect
Molecule", command=self.show_graph)
        self.mol_btn.grid(row=2, column=0, padx=20, pady=10)




        # Main content frame
        self.main_frame = ctk.CTkFrame(self)
        self.main_frame.grid(row=0, column=1, sticky="nswe")


        # Create the different screens as frames
        self.home_frame = MainScreen(self.main_frame)
        self.calculator_frame = CalculatorScreen(self.main_frame)
        self.molecule_frame = MoleculeScreen(self.main_frame)


        # Add content to the frames (example: just a label)


        self.show_home()  # show home screen by default
```

4.2: Ecranul “Home”

Ecranul “Home” reprezintă ecranul cu informații legate de aplicație:



Importuri:

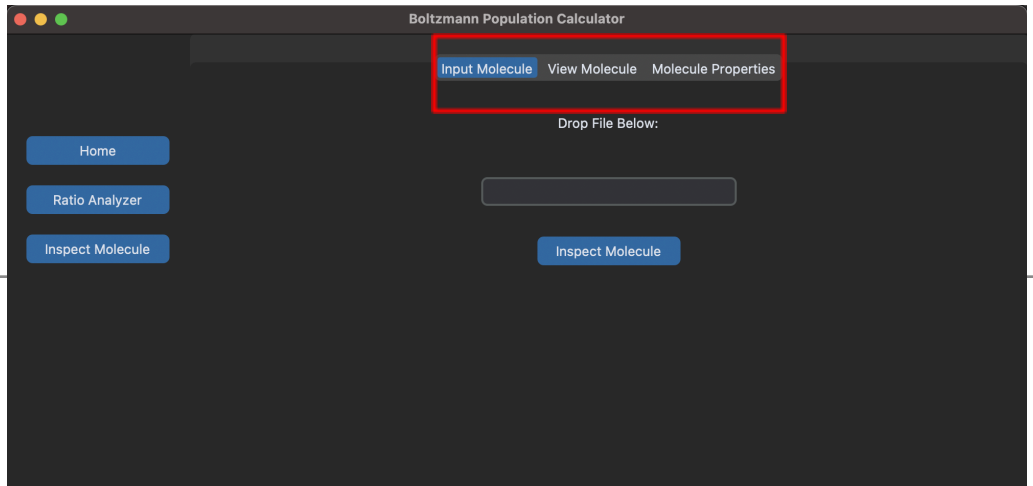
```
import custtkinter as ctk
```

Inițializarea ecranului:

```
class MainScreen(ctk.CTkFrame):

    def __init__(self, master):

        super().__init__(master)



        ctk.CTkLabel(self, text="Home Screen", font=("Arial",
30)).pack(pady=20)

        ctk.CTkLabel(self, text="Ratio Analyzer: ", font=("Arial",
20)).pack(pady=(60, 0), anchor='w')

        ctk.CTkLabel(self, text="The screen where you can input all
your desired molecule \n file (Gaussian .log file) and calculate the
Boltzmann population", font=("Arial", 15)).pack(pady=10, anchor='w')

        ctk.CTkLabel(self, text="Molecule Inspect: ", font=("Arial",
20)).pack(pady=(60, 0), anchor='w')

        ctk.CTkLabel(self, text="The screen where you can observe the
molecule that you \n input (Gaussian .log file) as well as analyze
different energy levels", font=("Arial", 15)).pack(pady=10,
anchor='w')
```

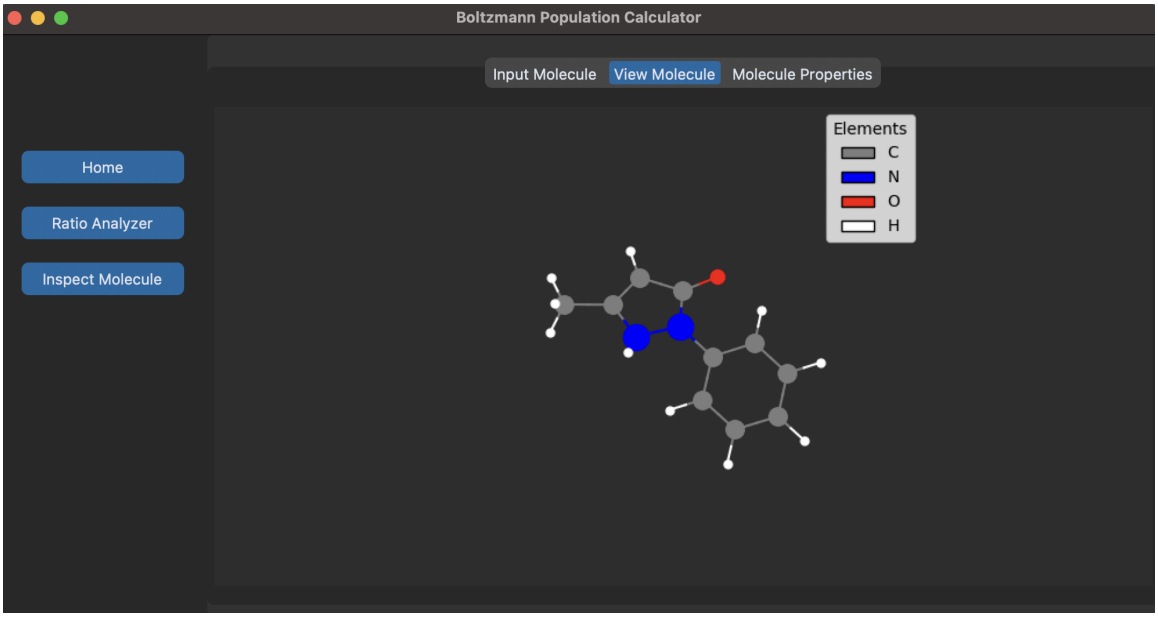
4.3 Ecranul “Inspect Molecule”

Acest ecran e configurat din mai multe ferestre pentru a structura informația mai ușor:

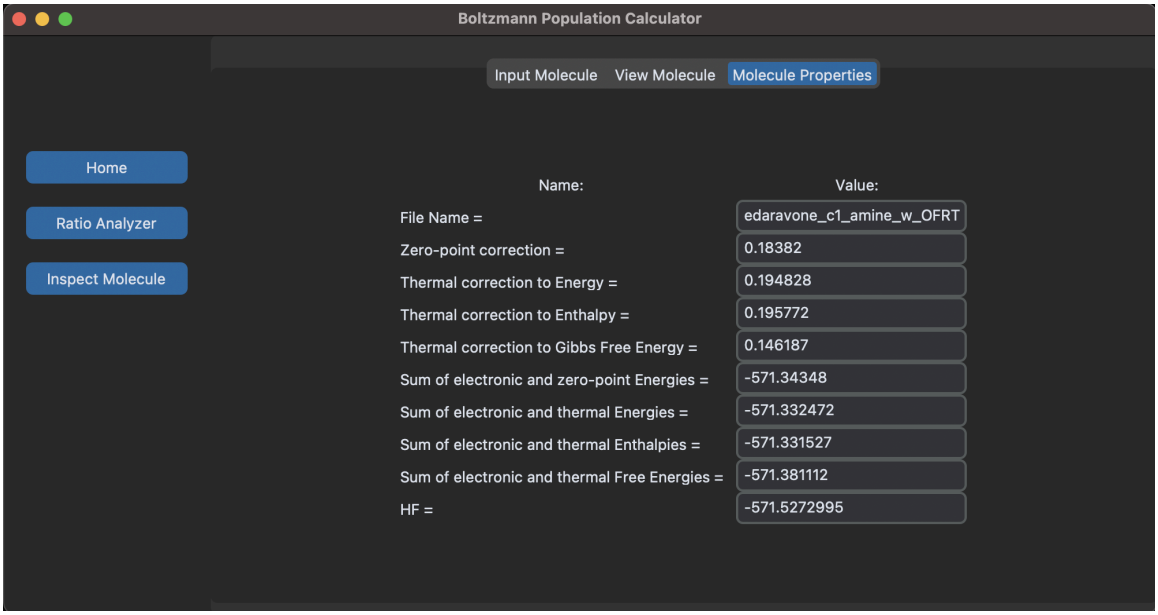


Fereastra “Input Molecule” constă dintr-un câmp pentru text și un buton ce dă start metodei de a extrage datele din fișier.

În fereastra “View Molecule” se crează un complot 3D pentru a putea vizualiza molecula:



Fereastra “Molecule Properties” va perezenta mereu informația legată de energiile moleculei, valorile cărora sunt afișate într-un câmp de text neiterabil:



Metodele utilizate:

Importuri:

```
from matplotlib.figure import Figure

import customtkinter as ctk

from matplotlib.patches import Patch

from mpl_toolkits.mplot3d import Axes3D

from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

from Backend.Loader.FileLoader import *

from tkinterdnd2 import DND_FILES
```

Funția pentru înregistrarea în câmp a fișierului prin metoda “Drag and Drop”:

```
def drop(self, event):
```

```
    self.entry_sv.set(event.data)
```

Funcția ce dă start execuției extragerii informației la apăsarea butonului:

```
def start_mol(self):
    self.draw_plot(self.tab_1, self.entry_sv.get())
    self.write_properties(self.entry_sv.get())
```

Metoda folosită pentru reprezentarea structurii moleculare:

```
def draw_plot(self, container, f):
    if self.canvas:
        self.canvas.get_tk_widget().destroy()
    if f == "":
        return
    self.figure = Figure(figsize=(10, 5), dpi=100)
    self.figure.subplots_adjust(left=0, right=1, top=1, bottom=0)
    ax = self.figure.add_subplot(111, projection='3d')


    colors = {'H': ["white", 1],
              'O': ["red", 3],
              'C': ["gray", 5],
              'N': ["blue", 10],
              'F': ["lightblue", 10],
              'S': ["yellow", 10],
              "B": ['pink', 10],
              "Cl": ["green", 10],
```

```
            "P": ["orange", 10]}


    atoms, bonds = load_molecule(f)
    xs = [float(atom[1][0]) for atom in atoms.values()]
    ys = [float(atom[1][1]) for atom in atoms.values()]
    zs = [float(atom[1][2]) for atom in atoms.values()]

    all_coords = xs + ys + zs
    min_val = min(all_coords)
    max_val = max(all_coords)

    ax.set_xlim(min_val, max_val)
    ax.set_ylim(min_val, max_val)
    ax.set_zlim(min_val, max_val)
    plotted_elems = set()
    for atom in atoms.values():
        ax.scatter(atom[1][0], atom[1][1], atom[1][2],
                   color=colors[atom[0]][0], s=25 *
colors[atom[0]][1])
        plotted_elems.add(atom[0])

    for bond in bonds:
        x_mid = (atoms[bond[0]][1][0] + atoms[bond[1]][1][0]) / 2
        y_mid = (atoms[bond[0]][1][1] + atoms[bond[1]][1][1]) / 2
        z_mid = (atoms[bond[0]][1][2] + atoms[bond[1]][1][2]) / 2
```

```
        ax.plot([atoms[bond[0]][1][0], x_mid],
                [atoms[bond[0]][1][1], y_mid],
                [atoms[bond[0]][1][2], z_mid],
                color=colors[atoms[bond[0]][0]][0])
        ax.plot([atoms[bond[1]][1][0], x_mid],
                [atoms[bond[1]][1][1], y_mid],
                [atoms[bond[1]][1][2], z_mid],
                color=colors[atoms[bond[1]][0]][0])


    # Plot the surface with a smooth colormap
    legend_elements = [
        Patch(facecolor=colors[element][0], edgecolor='black',
label=element)
        for element in plotted_elems
    ]


    ax.legend(handles=legend_elements, title="Elements", loc='upper
right', frameon=True)
    # Make the background dark
    ax.set_facecolor('#2e2e2e')  # Dark background
    self.figure.patch.set_facecolor('#2e2e2e')  # Make figure
background dark


    # Hide gridlines and panes
    ax.grid(False)
    ax.xaxis.pane.set_visible(False)
```

```
    ax.yaxis.pane.set_visible(False)
    ax.zaxis.pane.set_visible(False)
    ax.xaxis.line.set_visible(False)
    ax.yaxis.line.set_visible(False)
    ax.zaxis.line.set_visible(False)

    # Optional: Hide ticks and labels for ultra-clean look
    ax.set_xticks([])
    ax.set_yticks([])
    ax.set_zticks([])
    ax.set_xlabel('')
    ax.set_ylabel('')
    ax.set_zlabel('')

    # Draw and pack the canvas
    self.canvas = FigureCanvasTkAgg(self.figure, master=container)
    self.canvas.draw()
    widget = self.canvas.get_tk_widget()
    widget.pack(pady=10, fill="both", expand=True)
```

Metoda folosită pentru reprezentarea valorilor energiilor moleculei:

```
def write_properties(self, f):
    if f == "":
```

```
        return


    energies = load_properties(f)

    hf = load_hf(f)


    self.value0.configure(state="normal")  # allow editing

    self.value0.delete(0, "end")  # clear old text

     self.value0.insert(0,  f.split('/')[-1].strip('.log'))   # insert
new text

    self.value0.configure(state="readonly")


    self.value1.configure(state="normal")  # allow editing

    self.value1.delete(0, "end")  # clear old text

    self.value1.insert(0, energies[0])  # insert new text

    self.value1.configure(state="readonly")


    self.value2.configure(state="normal")  # allow editing

    self.value2.delete(0, "end")  # clear old text

    self.value2.insert(0, energies[1])  # insert new text

    self.value2.configure(state="readonly")


    self.value3.configure(state="normal")  # allow editing

    self.value3.delete(0, "end")  # clear old text

    self.value3.insert(0, energies[2])  # insert new text

    self.value3.configure(state="readonly")
```

```
        self.value4.configure(state="normal")  # allow editing
        self.value4.delete(0, "end")  # clear old text
        self.value4.insert(0, energies[3])  # insert new text
        self.value4.configure(state="readonly")

        self.value5.configure(state="normal")  # allow editing
        self.value5.delete(0, "end")  # clear old text
        self.value5.insert(0, energies[4])  # insert new text
        self.value5.configure(state="readonly")

        self.value6.configure(state="normal")  # allow editing
        self.value6.delete(0, "end")  # clear old text
        self.value6.insert(0, energies[5])  # insert new text
        self.value6.configure(state="readonly")

        self.value7.configure(state="normal")  # allow editing
        self.value7.delete(0, "end")  # clear old text
        self.value7.insert(0, energies[6])  # insert new text
        self.value7.configure(state="readonly")

        self.value8.configure(state="normal")  # allow editing
        self.value8.delete(0, "end")  # clear old text
        self.value8.insert(0, energies[7])  # insert new text
        self.value8.configure(state="readonly")
```

```
    self.value9.configure(state="normal")
    self.value9.delete(0, "end")
    self.value9.insert(0, hf)
    self.value9.configure(state="readonly")
```

Inițializarea ecranului “Inspect Molecule”:

```
class MoleculeScreen(ctk.CTkFrame):
    def __init__(self, master):
        super().__init__(master)

        self.canvas = None
        self.figure = None

        self.my_tab = ctk.CTkTabview(self)
        self.my_tab.pack(pady=10, fill="both", expand=True)
        self.tab_0 = self.my_tab.add("Input Molecule")
        self.tab_1 = self.my_tab.add("View Molecule")
        self.tab_2 = self.my_tab.add("Molecule Properties")
        ctk.CTkLabel(self.tab_0, text="Drop File
Below:").pack(pady=20)
        self.properties_frame = ctk.CTkFrame(self.tab_2)
        self.properties_frame.pack(expand=True)  # This will center
it by default

        self.entry_sv = ctk.StringVar()
```

```
        self.entry_sv.set("")

        self.entry = ctk.CTkEntry(self.tab_0,
textvariable=self.entry_sv, width=250)

        self.entry.pack(pady=20)

        self.entry.drop_target_register(DND_FILES)

        self.entry.dnd_bind("<<Drop>>", self.drop)



        # Table for the properties tab

        # Header of the table

        labelN = ctk.CTkLabel(self.properties_frame, text="Name:")

        labelN.grid(row=0, column=0, padx=(10, 5))



        labelV = ctk.CTkLabel(self.properties_frame, text="Value:")

        labelV.grid(row=0, column=1, padx=(10, 5))

        # Names of the Energies

        label0 = ctk.CTkLabel(self.properties_frame, text="File Name
=")

        label0.grid(row=1, column=0, padx=(10, 5), sticky='w')



        label1 = ctk.CTkLabel(self.properties_frame, text="Zero-point
correction =")

        label1.grid(row=2, column=0, padx=(10, 5), sticky='w')



        label2 = ctk.CTkLabel(self.properties_frame, text="Thermal
correction to Energy =")

        label2.grid(row=3, column=0, padx=(10, 5), sticky='w')
```

```
        label3 = ctk.CTkLabel(self.properties_frame, text="Thermal
correction to Enthalpy =")

        label3.grid(row=4, column=0, padx=(10, 5), sticky='w')


        label4 = ctk.CTkLabel(self.properties_frame, text="Thermal
correction to Gibbs Free Energy =")

        label4.grid(row=5, column=0, padx=(10, 5), sticky='w')


        label5 = ctk.CTkLabel(self.properties_frame, text="Sum of
electronic and zero-point Energies =")

        label5.grid(row=6, column=0, padx=(10, 5), sticky='w')


        label6 = ctk.CTkLabel(self.properties_frame, text="Sum of
electronic and thermal Energies =")

        label6.grid(row=7, column=0, padx=(10, 5), sticky='w')


        label7 = ctk.CTkLabel(self.properties_frame, text="Sum of
electronic and thermal Enthalpies =")

        label7.grid(row=8, column=0, padx=(10, 5), sticky='w')


        label8 = ctk.CTkLabel(self.properties_frame, text="Sum of
electronic and thermal Free Energies =")

        label8.grid(row=9, column=0, padx=(10, 5), sticky='w')


        label9 = ctk.CTkLabel(self.properties_frame, text="HF =")

        label9.grid(row=10, column=0, padx=(10, 5), sticky='w')
```

```
        # Entries for the values
        self.value0 = ctk.CTkEntry(self.properties_frame, width=200)
        self.value0.configure(state="readonly")
        self.value0.grid(row=1, column=1, padx=(5, 10))

        self.value1 = ctk.CTkEntry(self.properties_frame, width=200)
        self.value1.configure(state="readonly")
        self.value1.grid(row=2, column=1, padx=(5, 10))

        self.value2 = ctk.CTkEntry(self.properties_frame, width=200)
        self.value2.configure(state="readonly")
        self.value2.grid(row=3, column=1, padx=(5, 10))

        self.value3 = ctk.CTkEntry(self.properties_frame, width=200)
        self.value3.configure(state="readonly")
        self.value3.grid(row=4, column=1, padx=(5, 10))

        self.value4 = ctk.CTkEntry(self.properties_frame, width=200)
        self.value4.configure(state="readonly")
        self.value4.grid(row=5, column=1, padx=(5, 10))

        self.value5 = ctk.CTkEntry(self.properties_frame, width=200)
        self.value5.configure(state="readonly")
        self.value5.grid(row=6, column=1, padx=(5, 10))
```

```
        self.value6 = ctk.CTkEntry(self.properties_frame, width=200)
        self.value6.configure(state="readonly")
        self.value6.grid(row=7, column=1, padx=(5, 10))

        self.value7 = ctk.CTkEntry(self.properties_frame, width=200)
        self.value7.configure(state="readonly")
        self.value7.grid(row=8, column=1, padx=(5, 10))

        self.value8 = ctk.CTkEntry(self.properties_frame, width=200)
        self.value8.configure(state="readonly")
        self.value8.grid(row=9, column=1, padx=(5, 10))

        self.value9 = ctk.CTkEntry(self.properties_frame, width=200)
        self.value9.configure(state="readonly")
        self.value9.grid(row=10, column=1, padx=(5, 10))
        # Button for the start of file loading
        ctk.CTkButton(self.tab_0, text="Inspect Molecule",
command=self.start_mol).pack(pady=10)
```

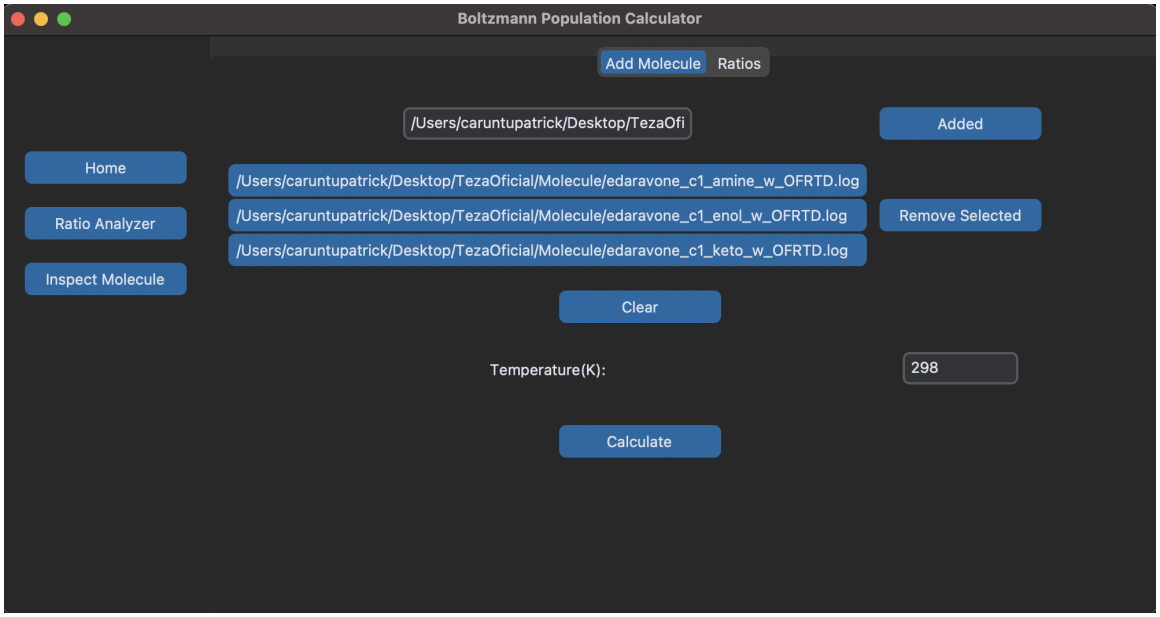
4.4 Ecranul "Ratio Analyzer"

La fel ca și ecranul precedent, acesta folosește ferestre diferite pentru a reprezenta informația.

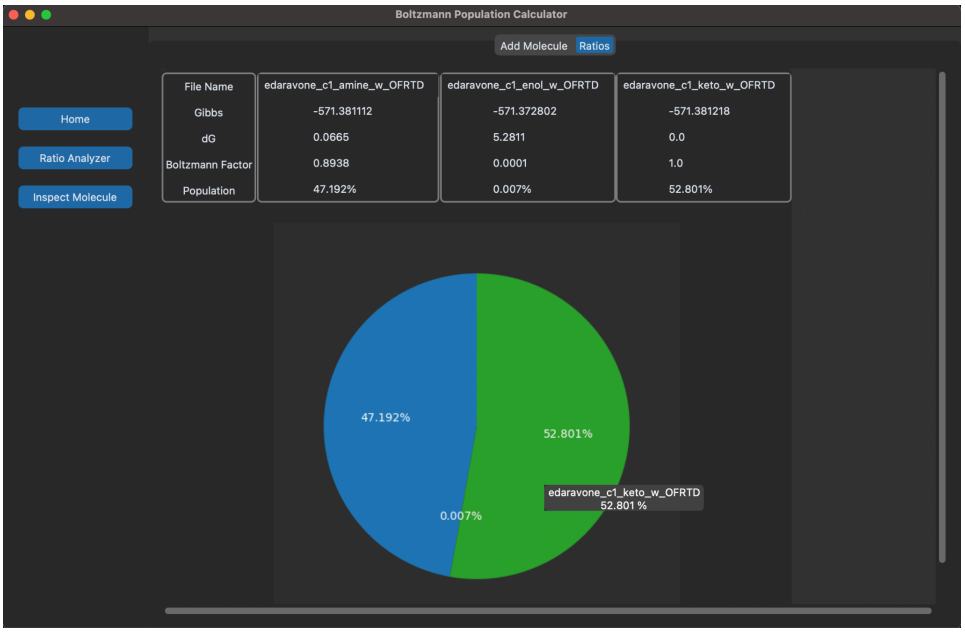
În prima fereastră se include de către utilizator informația necesară pentru a începe calculul:

- Câmp de text pentru inserarea fișierelor
- Buton pentru adăugare
- Un câmp pentru afișarea moleculelor din listă

- Un buton pentru ștergere a unei molecule
- Un buton pentru ștergere a listei
- Un câmp de text pentru setarea temperaturii
- Un buton pentru a executa calculul



A doua fereastră inițial e goală, dar la executarea calculului aceasta împarte pagina și va prezenta structurat informația sub forma unui tabel și a unei diagrame circulare interacționabile.



Metode folosite:

Importuri:

```
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
import customtkinter as ctk
import matplotlib.pyplot as plt
import numpy as np
from Backend.Functions.Functions import *
from tkinterdnd2 import DND_FILES
from App.App import *
from Backend.Loader.FileLoader import *
from App.Screens.MoleculeScreen import *
import os
```

Funcția pentru înregistrarea fișierelor prin metoda “Drag and Drop”:

```
def drop(self, event):

    self.entry_sv.set(event.data)
```

Funcția pentru actualizarea listei de molecule:

```
def update_molecule_list(self):

    for widget in self.molecule_frame.winfo_children():

        widget.destroy()

    for mol in sorted(self.molecules):

        item = ctk.CTkButton(self.molecule_frame, text=mol,
width=250, anchor="w",

                             command=lambda m=mol:
self.select_molecule(m))

        item.molecule = mol

        item.pack(fill="x", pady=1)
```

Funcția pentru adăugarea a moleculelor noi prin căutarea într-un director sau introducerea fișierelor:

```
def add_molecule(self):

    try:

        dir = self.entry_sv.get()

        new_molecules = os.listdir(dir)

        for i in range(len(new_molecules)):

            new_molecules[i] = dir + '/' + new_molecules[i]

    except FileNotFoundError and NotADirectoryError:

        new_molecules = self.entry_sv.get().split()
```

```
    for new_molecule in new_molecules:
        if new_molecule and new_molecule not in self.molecules:
            self.molecules.add(new_molecule)
    self.update_molecule_list()
    self.adder.configure(text="Added")
```

Funcția pentru selectarea unei molecule din listă:

```
def select_molecule(self, molecule):
    self.selected_molecule = molecule
    for child in self.molecule_frame.winfo_children():
        if hasattr(child, "molecule"):
            if child.molecule == molecule:
                child.configure(fg_color="#444444")
            else:
                child.configure(fg_color="transparent")
```

Funcția pentru ștergerea moleculei selectate din listă:

```
def remove_selected(self):
    if self.selected_molecule in self.molecules:
        self.molecules.remove(self.selected_molecule)
        self.selected_molecule = None
        self.update_molecule_list()
```

Funcția pentru ștergerea completă a listei de molecule:

```
def clear_all(self):
    self.molecules.clear()
    self.update_molecule_list()
    for widget in self.results_frame.winfo_children():
        widget.destroy()
    self.wedges_info.clear()
    if self.tooltip:
        self.tooltip.destroy()
        self.tooltip = None
```

Funcția pentru detectarea de către aplicație a utilizarii roții mouse-ului:

```
def _on_mousewheel(self, event):
    try:
        if event.state & 0x0001:  # Shift key held
            direction = -1 if event.delta > 0 else 1
            self.canvas.xview_scroll(direction, "units")
        elif event.num == 4:
            self.canvas.yview_scroll(-1, "units")
        elif event.num == 5:
            self.canvas.yview_scroll(1, "units")
        else:
            direction = -1 if event.delta > 0 else 1
            self.canvas.yview_scroll(direction, "units")
    except:
        pass
```

Funcția ce permite parcurgerea cu ajutorul roții mouse-ului a ferestrei:

```
def _bind_mousewheel_recursive(self, widget):
    widget.bind("<MouseWheel>", self._on_mousewheel)
    widget.bind("<Button-4>", self._on_mousewheel)
    widget.bind("<Button-5>", self._on_mousewheel)
    for child in widget.winfo_children():
        self._bind_mousewheel_recursive(child)
```

Funcția ce afișează informația despre diagramă când aplicația detectează că mouse-ul planează deasupra acesteia:

```
def _mpl_hover(self, event):
    if not self.tooltip:
        self.tooltip = ctk.CTkLabel(self.results_frame, text="",
corner_radius=5,
                                    fg_color="#444",
text_color="white")
        self.tooltip.place_forget()

    if event.inaxes:
        for wedge, label, pop in self.wedges_info:
            if wedge.contains_point((event.x, event.y)):
                self.tooltip.configure(text=f"{label}\n{pop:.3f} %")
                # Update tooltip position on every motion
                x = int(event.guiEvent.x_root -
self.results_frame.winfo_rootx() + 10)
```

```
                y = int(event.guiEvent.y_root -
self.results_frame.winfo_rooty() + 10)

                self.tooltip.place(x=x, y=y)

                return


    self.tooltip.place_forget()
```

Funcția pentru afișarea rezultatelor calculului Boltzmann și afișarea acestora în pagină:

```
def calculate(self):

    for widget in self.results_frame.winfo_children():

        widget.destroy()

    self.wedges_info.clear()

    if self.tooltip:

        self.tooltip.destroy()

        self.tooltip = None


    cell = ctk.CTkFrame(self.results_frame, fg_color="transparent",
border_width=2, border_color='gray')

    cell.grid(row=0, column=0, rowspan=2, pady=5, sticky="n")

    ctk.CTkLabel(cell, text="File Name", anchor="w").pack(padx=5,
pady=2)

    ctk.CTkLabel(cell, text="Gibbs", anchor="w").pack(padx=5, pady=2)

    ctk.CTkLabel(cell, text="dG", anchor="w").pack(padx=5, pady=2)

    ctk.CTkLabel(cell, text="Boltzmann Factor",
anchor="w").pack(padx=5, pady=2)

    ctk.CTkLabel(cell, text="Population", anchor="w").pack(padx=5,
pady=2)
```

```
    gibbs = []
    filenames = []
    for mol in self.molecules:
        filename = mol.split('/')[-1].replace('.log', '')
        filenames.append(filename)
        energies = load_properties(mol)
        gibbs.append(energies[7])

    deltas, factors, population = population_calc(gibbs,
float(self.temperature.get()))
    for i in range(len(gibbs)):
        filename = filenames[i]
        energy_str = str(gibbs[i])
        delta_str = str(round(deltas[i], 4))
        factor_str = str(round(factors[i], 4))
        population_str = str(population[i]) + '%'
        filename_width = max(10, len(filename))

        cell = ctk.CTkFrame(self.results_frame,
fg_color="transparent", border_width=2, border_color='gray')
        cell.grid(row=0, column=i + 1, rowspan=2, pady=5, sticky="n")

        ctk.CTkEntry(cell, corner_radius=0,
textvariable=ctk.StringVar(value=filename),
                     state='readonly', fg_color="transparent",
border_width=0,
```

```
                    width=int(filename_width * 8.5)).pack(pady=2,
padx=2)



        ctk.CTkEntry(cell, corner_radius=0,
textvariable=ctk.StringVar(value=energy_str),

                    state='readonly', fg_color="transparent",
border_width=0,

                    width=100).pack(pady=2, padx=2)



        ctk.CTkEntry(cell, corner_radius=0,
textvariable=ctk.StringVar(value=delta_str),

                    state='readonly', fg_color="transparent",
border_width=0,

                    width=100).pack(pady=2, padx=2)



        ctk.CTkEntry(cell, corner_radius=0,
textvariable=ctk.StringVar(value=factor_str),

                    state='readonly', fg_color="transparent",
border_width=0,

                    width=100).pack(pady=2, padx=2)



        ctk.CTkEntry(cell, corner_radius=0,
textvariable=ctk.StringVar(value=population_str),

                    state='readonly', fg_color="transparent",
border_width=0,

                    width=100).pack(pady=2, padx=2)



    y = np.array(population)

    fig = Figure(figsize=(5, 5), dpi=100, facecolor='#2e2e2e')
```

```
    ax = fig.add_subplot(111)


    wedges, texts, autotexts = ax.pie(
        y,
        autopct='%1.3f%%',
        textprops={'color': 'w'},
        startangle=90,
        counterclock=True
    )


    self.figure = fig
    self.wedges_info = list(zip(wedges, filenames, population))


    fig.tight_layout()


    self.plot_canvas = FigureCanvasTkAgg(fig,
master=self.results_frame)
    self.plot_canvas.draw()
    plot_widget = self.plot_canvas.get_tk_widget()
    plot_widget.grid(row=2, column=0, columnspan=len(filenames) + 1,
pady=20)
    self.plot_canvas.mpl_connect("motion_notify_event",
self._mpl_hover)




    self._bind_mousewheel_recursive(self.results_frame)
```

Inițializarea ecranului “Ratio Analyzer”:

```
class CalculatorScreen(ctk.CTkFrame):

   def __init__(self, master):

       super().__init__(master)


       self.molecules = set()

       self.figure = None

       self.tooltip = None

       self.canvas = None

       self.wedges_info = []

       self.selected_molecule = None


       # Main tabview

       self.my_tab = ctk.CTkTabview(self)

       self.my_tab.pack(fill="both", expand=True)

       self.tab_a = self.my_tab.add("Add Molecule")

       self.tab_b = self.my_tab.add("Ratios")


       #table_frame container


       self.table_frame = ctk.CTkFrame(self.tab_a)

       self.table_frame.pack(fill="both", expand=True, padx=10,
pady=10)
```

```
        # Entry field

        self.entry_sv = ctk.StringVar()

        self.entry = ctk.CTkEntry(self.table_frame,
textvariable=self.entry_sv, width=250)

        self.entry.grid(row=0, column=0, pady=10)

        self.entry.drop_target_register(DND_FILES)

        self.entry.dnd_bind("<<Drop>>", self.drop)



        # Add button

        self.adder = ctk.CTkButton(self.table_frame, text='Add',
command=self.add_molecule)

        self.adder.grid(row=0, column=1, padx=10)



        # Molecule item list

        self.molecule_frame = ctk.CTkFrame(self.table_frame,
fg_color="transparent")

        self.molecule_frame.grid(row=1, column=0, pady=10)

        self.update_molecule_list()



        ctk.CTkButton(self.table_frame, text='Remove Selected',
command=self.remove_selected).grid(row=1, column=1, padx=10)

        ctk.CTkButton(self.table_frame, text='Clear',
command=self.clear_all).grid(row=2, columnspan=2, pady=10)



        self.temperature = ctk.StringVar()

        self.temperature.set('298')
```

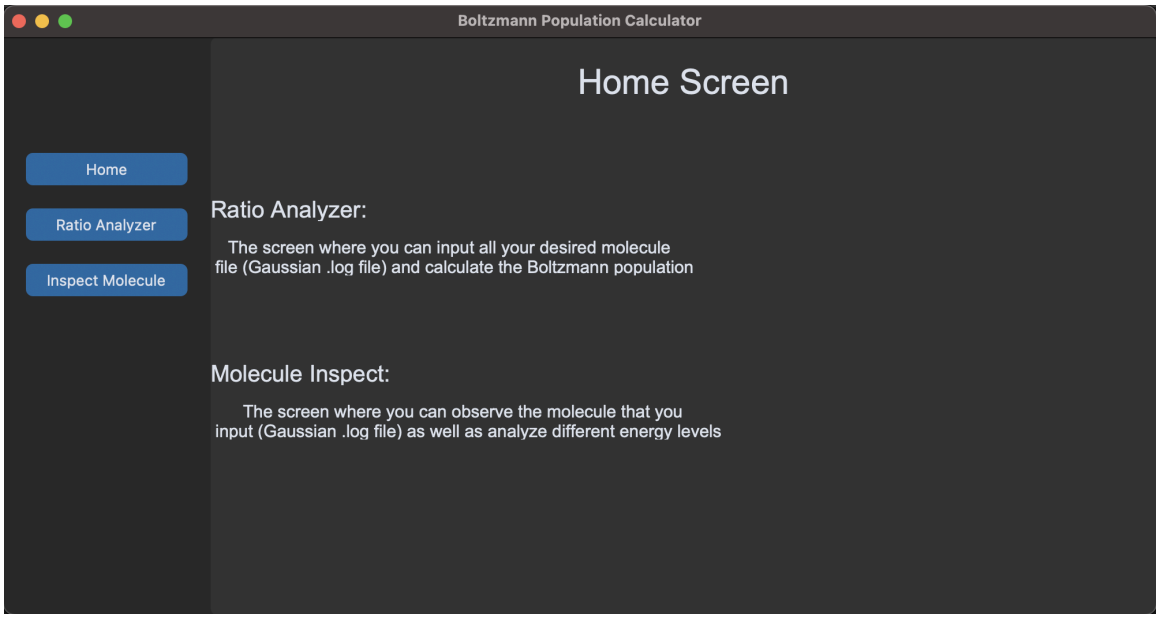
```
        ctk.CTkLabel(self.table_frame,
text='Temperature(K):').grid(row=3, column=0, pady=15)

        self.temp_entry = ctk.CTkEntry(self.table_frame,
textvariable=self.temperature, width=100).grid(row=3, column=1,
pady=15)



        ctk.CTkButton(self.table_frame, text='Calculate',
command=self.calculate).grid(row=4, columnspan=2, pady=20)



        # Scrollable results container

        self.scroll_container = ctk.CTkFrame(self.tab_b)

        self.scroll_container.pack(fill="both", expand=True, padx=10,
pady=10)



        self.canvas = ctk.CTkCanvas(self.scroll_container,
borderwidth=0, highlightthickness=0)

        self.v_scrollbar = ctk.CTkScrollbar(self.scroll_container,
orientation="vertical", command=self.canvas.yview)

        self.h_scrollbar = ctk.CTkScrollbar(self.scroll_container,
orientation="horizontal", command=self.canvas.xview)



        self.canvas.configure(yscrollcommand=self.v_scrollbar.set,
xscrollcommand=self.h_scrollbar.set)



        self.canvas.grid(row=0, column=0, sticky="nsew")

        self.v_scrollbar.grid(row=0, column=1, sticky="ns")

        self.h_scrollbar.grid(row=1, column=0, sticky="ew")



        self.scroll_container.grid_rowconfigure(0, weight=1)
```

```
        self.scroll_container.grid_columnconfigure(0, weight=1)



        # Frame inside canvas for scrollable content

        self.results_frame = ctk.CTkFrame(self.canvas)

        self.canvas_window = self.canvas.create_window((0, 0),
window=self.results_frame, anchor="nw")

        self.results_frame.bind("<Configure>", lambda e:
self.canvas.configure(scrollregion=self.canvas.bbox("all")))
```

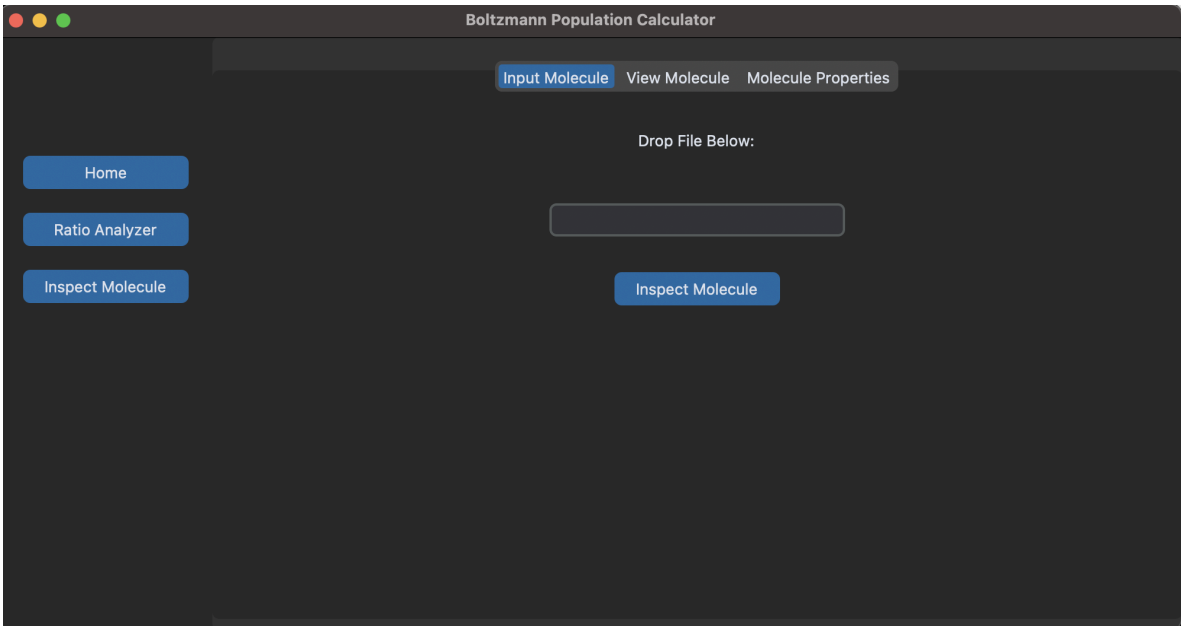
5. Utilizarea Aplicației

În acest capitol se va discuta despre navigarea aplicației și indicațiile pentru uitilizator pentru folosirea aplicației.

Odată ce se deschide aplicația, aceasta va afișa următorul ecran:

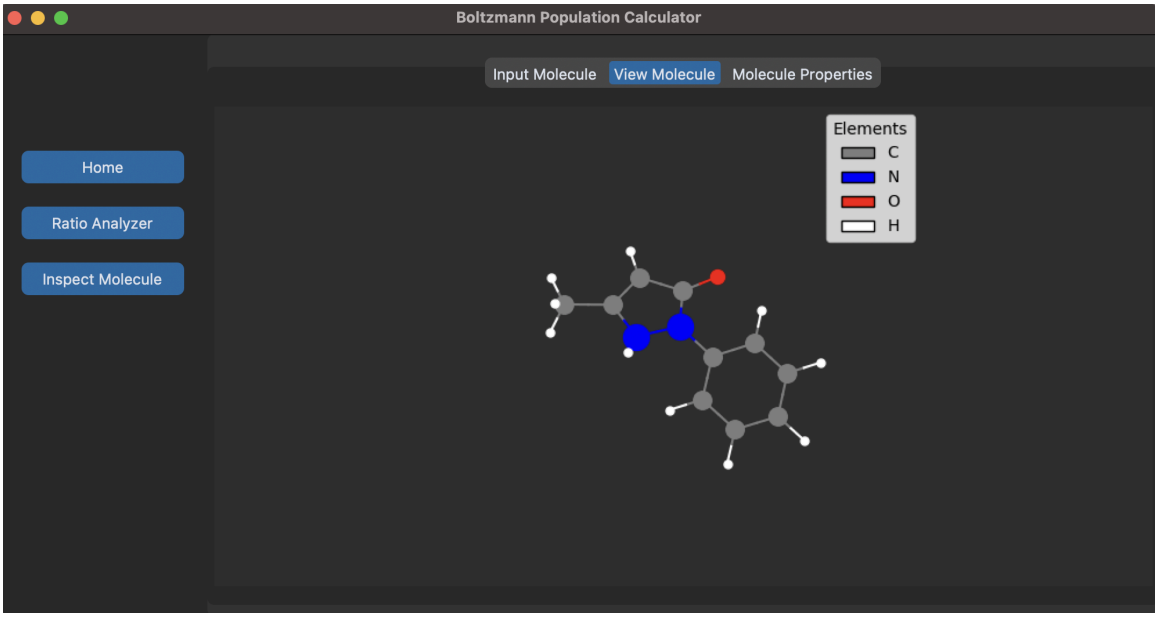


Aici pot fi găsite informații legate de butoanele din ecranul de navigare. Pentru a accesa alte ecrane se folosesc butoanele din partea stânga (ecranul de navigare). După apăsarea butonului “Inspect Molecule”, va fi prezentat următorul ecran:

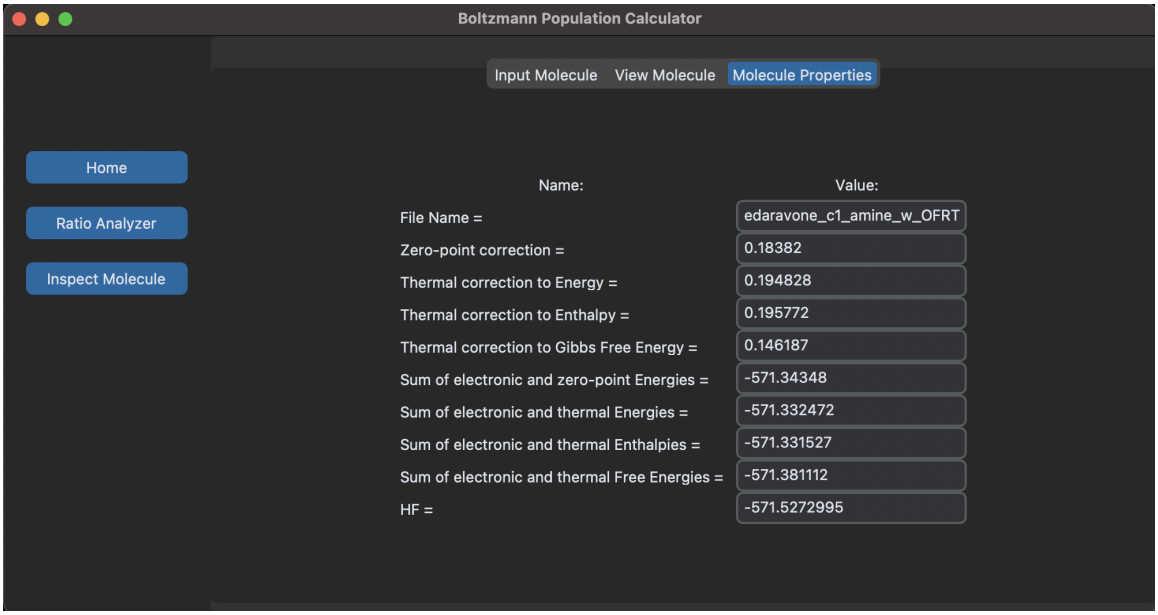


Aici puteți introduce prin “Drag and Drop” sau prin inserarea căii, un fișier de ieșire Gaussian (se introduce doar un fișier). La apăsarea butonului “Inspect molecule” din ecranul de afișare a informației, aplicația extrage informația despre moleculă și le afișează în tab-urile “View Molecule” și “Molecule Properties”.

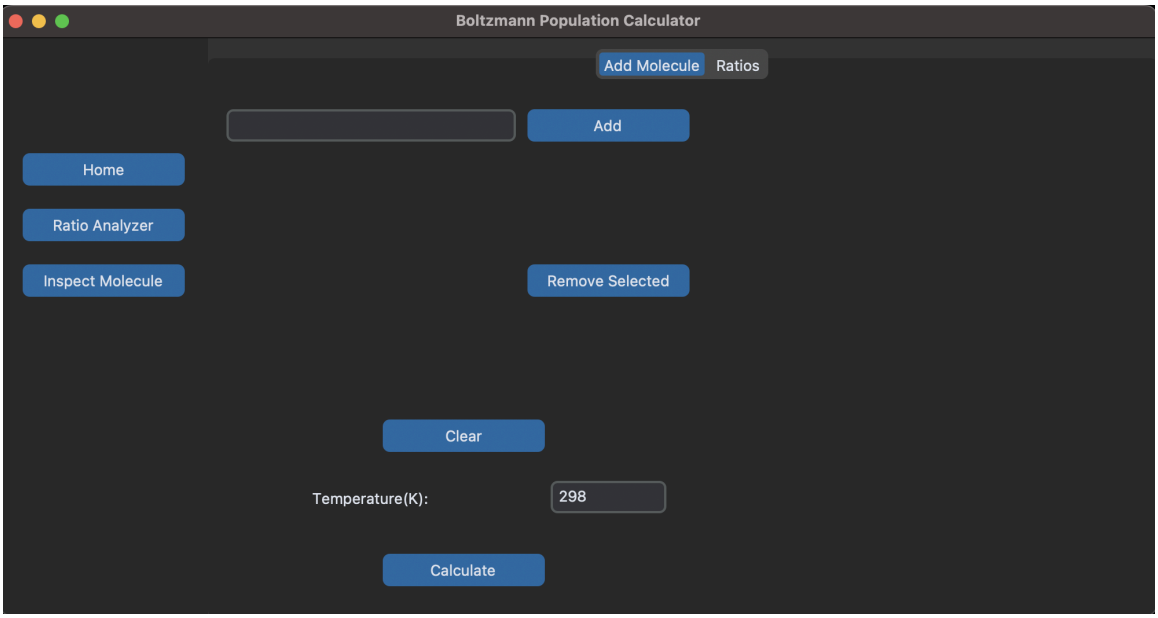
În “View Molecule” va fi prezentată structura moleculară unde se poate interacționa cu molecula (rotire, micșorare/mărire):



În “Molecule Properties” va fi descrisă informația legată de energiile moleculei:

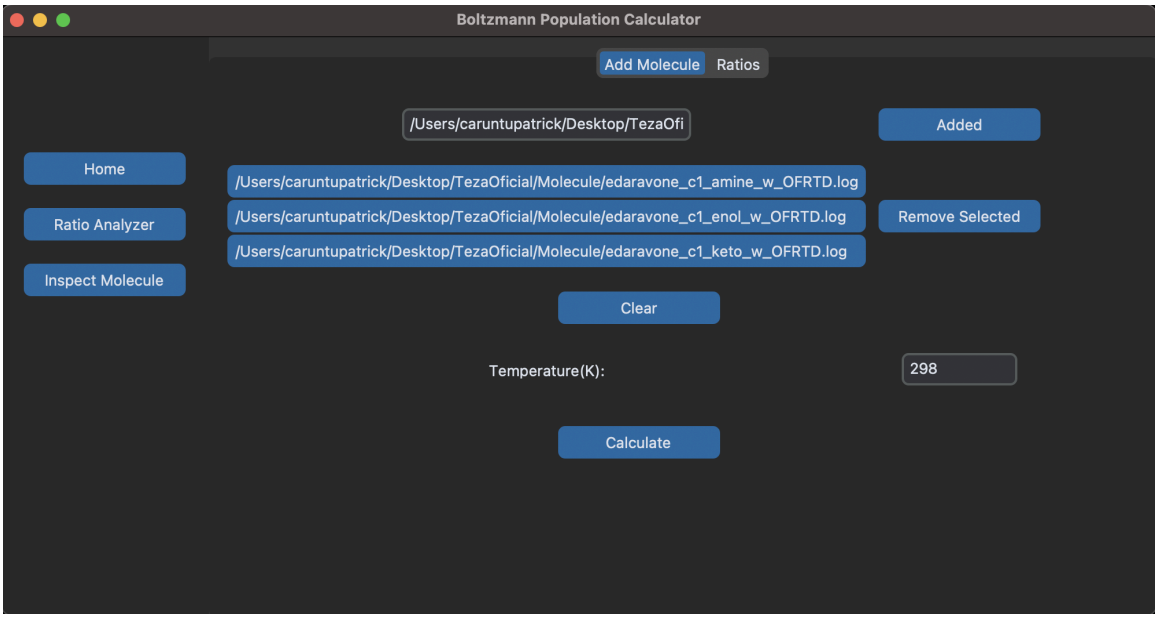


La apăsarea butonului “Ratio Analyzer” va fi afișat următorul ecran:



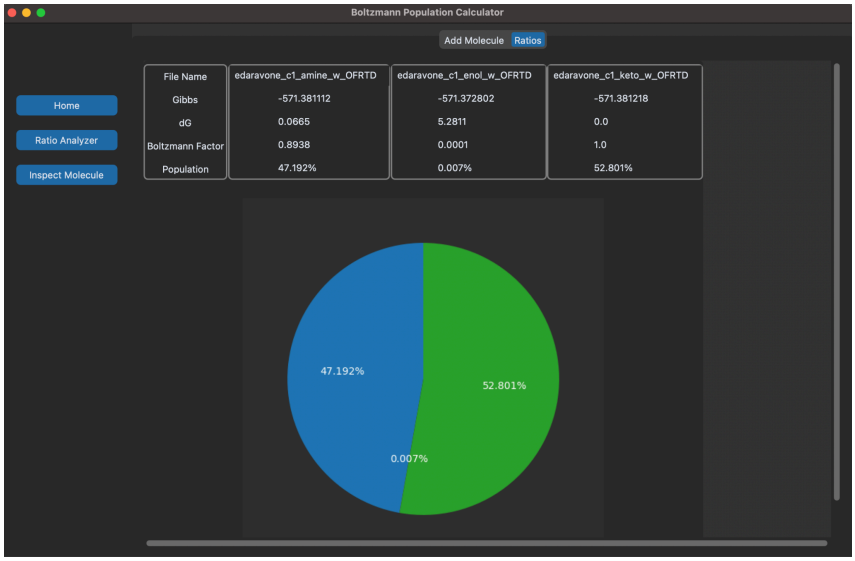
În prima căsuță se pot adăuga mai multe molecule prin “Drag and Drop” sau prin inserarea căii către fișierul de ieșire Gaussian. Pe langă inserarea unui singur fișier, aici este permisă adaugarea mai multor fișiere în același timp prin selectarea concomitentă și inserarea lor prin “Drag and Drop”. Este permisă și insera unui fișier de tip “Directory” din care aplicația va extrage toate fișierele din interiorul acestuia. Aplicația nu va permite inserarea aceluiași fișier de mai multe ori pentru evitarea erorilor.

După adăugarea mai multor fișiere, acestea vor fi salvate în memoria aplicației și prezentate sub formă de mai multe butoane:

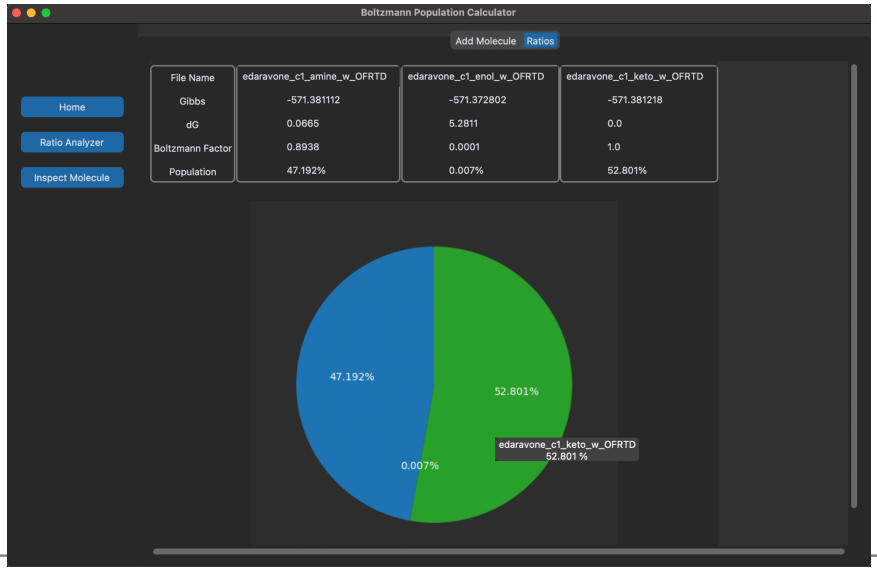


La apăsarea butonului “Clear”, toate moleculele din memoria aplicației vor fi șterse. Pentru a șterge un singur fișier, este nevoie de a selecta fișierul după care se apasă butonul “Remove Selected”.

Se alege temperatura (în Kelvin) necesară pentru calculul dorit al populației Boltzmann (valoare implicită este setată la 298 K), iar când totul este introdus, se apasă butonul “Calculate” și aplicația execută toate calculele și le afișează în fereastra “Ratios” sub forma unui tabel și al unui Pie-Chart::



Această fereastra permite și posibilitatea de a face “Scroll” pentru a parcurge toată informația. La planarea deasupra diagramei, aplicația va afișa informația despre regiunea dată



Concluzii

Lucrarea de față a urmărit dezvoltarea unei aplicații software pentru automatizarea calculului populațiilor Boltzmann, un instrument esențial pentru determinarea celor mai stabili conformeri sau tautomeri moleculari și pentru în analiza comportamentului termic al acestora.

Am pornit de la fundamentele teoretice ale distribuției Boltzmann, subliniind rolul său central în fizica statistică și în interpretarea fenomenelor spectroscopice. Am prezentat o arhitectură software clar structurată, dezvoltată în Python, ce permite extragerea datelor energetice din fișierele Gaussian și calculul automat al populațiilor relative pentru o gamă de temperaturi.

Aplicația include o interfață grafică intuitivă și funcționalități extinse: conversie de unități, manipulare de degenerescențe, analiză multi-temperatură și exportul rezultatelor. Prin studii de caz, am demonstrat atât corectitudinea implementării cât și utilitatea practică a aplicației în contexte educaționale și de cercetare.

În perspectivă, aplicația poate fi extinsă pentru a include suport pentru alți algoritmi statistici, suport pentru mai multe tipuri de fișiere de intrare și integrarea directă a interpretării spectroscopice. Astfel, instrumentul dezvoltat devine un exemplu elocvent al modului în care metodele computaționale moderne pot eficientiza și îmbunătăți procesele de analiză științifică.

Bibliografie

[Atk21] P. Atkins, J. de Paula, *Physical Chemistry*, Oxford University Press.

[CustTk] Tom Schimansky, *CustomTkinter Documentation*, https://github.com/TomSchimansky/CustomTkinter (accesat: 20.06.2025)

[Gau16] Gaussian Inc., *Gaussian 16 User Reference*, https://gaussian.com/ (accesat: 20.06.2025)

[Lev05] I. N. Levine, *Quantum Chemistry*, Pearson Education, 2005.

[Li19] X. Li et al., Near-infrared fluorophores for biomedical imaging, *Nature Reviews Materials*, 4 (2019) 679–701. DOI: https://doi.org/10.1038/s41578-019-0122-1

[Matp] Matplotlib Developers, *Matplotlib Documentation*, https://matplotlib.org (accesat: 20.06.2025)

[McQ00] D. A. McQuarrie, *Statistical Mechanics*, University Science Books, 2000.

[Nump] NumPy Developers, *NumPy Documentation*, https://numpy.org (accesat: 20.06.2025)

[PytDoc] Python Software Foundation, *Python 3 Documentation*, https://www.python.org (accesat: 20.06.2025)

[Yan08] J. Yang et al., Seminaphthofluorones are a family of water-soluble, low molecular weight, NIR-emitting fluorophores, *PNAS*, 105 (2008) 8829–8834. DOI: https://doi.org/10.1073/pnas.0803730105

Anexa 2

DECLARAŢIE PE PROPRIE RĂSPUNDERE

Subsemnatul, Căruntu Patrick, declar că Lucrarea de absolvire/Lucrarea de licență/Proiectul de diplomă/Lucrarea de disertaţie pe care o voi prezenta în cadrul examenului de finalizare a studiilor la Facultatea de Fizică din cadrul Universităţii Babeş-Bolyai, în sesiunea iunie 2025, sub îndrumarea Prof.dr. Vasile Chiș reprezintă o operă personală. Menționez că nu am plagiat o altă lucrare publicată, prezentată public sau un fişier postat pe Internet. Pentru realizarea lucrării am folosit exclusiv bibliografia prezentată şi nu am ascuns nici o altă sursă bibliografică sau fişier electronic pe care să le fi folosit la redactarea lucrării.

Prezenta declarație este parte a lucrării şi se anexează la aceasta.

Data,

Căruntu Patrick,

25.06.2025