Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Zentrum für Bioinformatik (ZBH)

Universität Hamburg

Hamburg, Deutschland

Projekt Strukturelle Bioinformatik

EFFIZIENTE MODELLIERUNG MOLEKULARER INTERAKTIONEN - DOKUMENTATION

Eingereicht von	Cindy Ching, Wang Mengdi, Kim Weisser, Asmaa Achoukhi, Andreas Blank
Datum	28.02.2017
Betreut von	Prof. Dr. Tobias Schwabe

Inhalt

1 Berechnung der Bindungsenergie mit emmi

Mit dem Programm 'emmi' (**E**ffiziente **M**odellierung **M**olekularer **I**nteraktionen) werden folgende Bindungsenergien zwischen Molekülen näherungsweise berechnet berücksichtigt:

- Dispersionsenergien
- Energien durch Wasserstoffbrücken
- Elektrostatische Energien
- Polarisierungsenergien
- Lösungsmittelenergien

Bei der Implementierung des Problems wurde sich für die Programmiersprache Python entschieden.

Das Hauptprogramm ruft alle 5 Teile nacheinander auf und addiert deren Energien. Die einzelnen Teile sind dabei in jeweils einer eigenen Datei implementiert, wobei das Hauptprogramm 'mi' die entsprechende Funktion importiert.
Die Funktionen für die Lösungsmittelenergie sowie für die Polarisierungsenergie liefert dabei direkt den gewünschten Beitrag. Für die anderen drei Beiträge wird der Energiebeitrag für den Gesamtkomplex berechnet und die Beiträge der einzelnen Moleküle abgezogen. Die Programmstruktur ist in Abbildung 1 dargestellt.

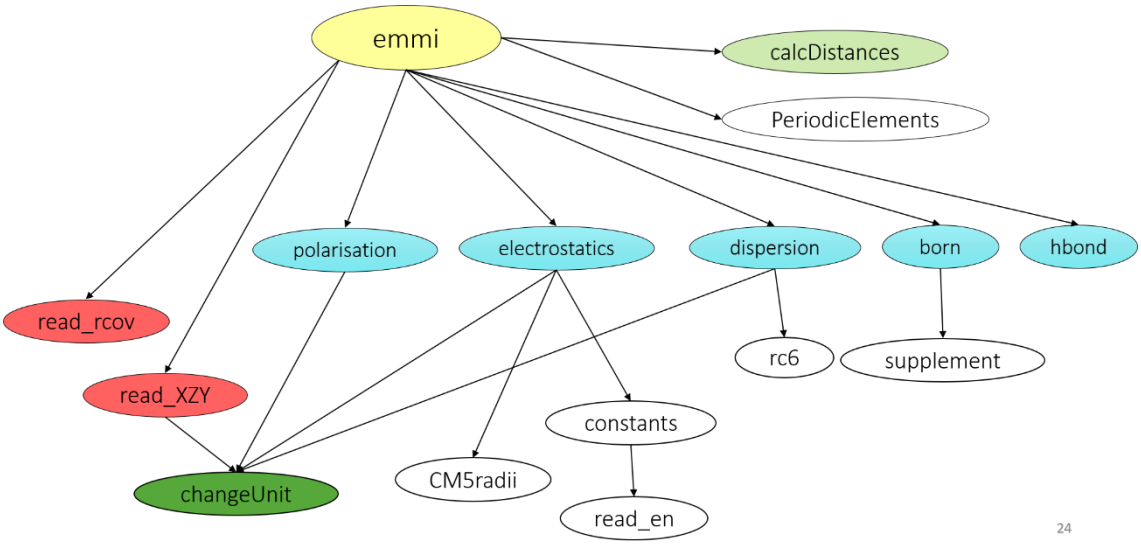

Abbildung 1: Programmstruktur von emmi.

Die Eingabedatei liefert folgende Werte:

- Anzahl der Atome im Komplex
- Koordinaten der Atome in Angstrom
- Typ des jeweiligen Atoms
- Molekülzugehörigkeit des Jeweiligen Atoms
- Nachbarn eines Atoms

Zusätzlich werden aus einer der Datei rcov.dat die Kovalenzradien einzelner Atomtypen eingelesen.

Insgesamt werden folgende Eingabeinformationen benötigt:

- Atomtypen: [String]: Elementsymbol (Länge: Anzahl der Atome im Molekül)
  - (Aus Eingabedatei)
- Atompositionen: [[float]]: x,y,z-Koordinaten der Atome in Atomareneinheiten
  - (Aus Eingabedatei)
- Anzahl der Atome int
  - (Aus Eingabedatei)
- Molekülzugehörigkeit [int]: Nummer des jeweiligen Moleküls (Länge: Anzahl der Atome im Molekül)
  - (Aus Eingabedatei)
- Distanzen zwischen den einzelnen Atomen: [float]
  - (Mit Atompositionen berechnet)
- Kovalenzradien der Atome [float]: entsprechend der Elemente im Periodensystem
  - (Aus rcov.dat eingelesen)
- Elemente des Periodensystems [String]
  - (statisch hinterlegt)
- Dispersionskoeffizienten: [float]: (Länge: Anzahl der Atome im Molekül)
  - (durch den Teil welcher die Dispersionsenergie berechnet geliefert)
- Partialladungen: [float]
  - (mit der Anzahl der Atome, Atomtypen, den Distanzen zwischen den Atomen, den Elementen sowie der Option ob mit oder ohne CM5-Korrektur berechnet)
- Koordinationszahlen: [float]
  - (mit der Anzahl der Atome, Atomtypen, den Distanzen zwischen den Atomen, den Kovalenzradien und den Elementen berechnet)
- Nachbarn eines Atoms

2 Unterfunktion zur Berechnung der Bindungsenergie zweier Moleküle

2.1 H-Brücken-Korrektur

calcHbonds() sucht mit Hilfe von 3 geschachtelten for-Schleifen nach einem möglichen Konstrukt, bestehend aus zwei elektronegativen Atomen N oder O und einem Wasserstoffatom, welches mit einem der Moleküle verbunden ist.
Es werden somit Atompaare bestehend aus atom_1 (N,O) atom_2 (N,O) und atom_3 (H) gefunden. In dieser Funktion erfolgt der Aufruf aller anderen Funktionen, zur Berechnung von Distanzen, Winkeln und letztlich der Energie für jedes Konstrukt. Wobei nur Energieterme von Konstrukten aufsummiert werden, bei denen cos_theta > 0 ergibt (vgl. Referenzcode Zeile 797). Die Funktion liefert den Gesamt-Energiewert in kcal/mol zurück.

calcDistOfPair() bestimmt die Distanz dieser drei Atome (atom_1 (N,O), atom_2 (N,O), atom_3 (H)) und speichert die Distanz zwischen atom_1 und atom_2 in dist_ab ab. In dist_xh wird die kürzere Distanz abgespeichert. Entweder die Distanz zwischen atom_1 und atom_3 oder die Distanz zwischen atom_2 und atom_3. dist_ab und dist_xh wird zurückgeliefert.

getNextNeighbour() bestimmt jeweils für atom_1(N,O) und atom_2(N,O) die nächsten Nachbarn. Mit Hilfe der Nachbar-Matrix und der Distanzmatrix kann dies ermittelt werden. r_a1, r_a2 und r_a3 bilden aufsteigend sortiert die nächsten Nachbarn. Das heißt Atom r_a1 ist der nächste Nachbar von atom_1.
Atom r_a2 ist der zweit-nächste Nachbar. Gleiches gilt für atom_2.

calcAngle() berechnet die Winkel Theta, Phi, und Alpha und liefert diese zurück.
CalcAngle wurde mit Hilfe des Referenzcodes Zeile 82 implementiert.

calcTorsion() bekommt die berechneten Winkel phi, alpha, und die Nachbarn eines jeden elektronegativen Atoms. Der Torsionswinkel wird mittels Formel: https://www.gdch.de/fileadmin/downloads/Netzwerk_und_Strukturen/Fachgruppen/Analytische_Chemie/chemkrist/jones_1.pdf
berechnet. Die Funktion liefert die berechneten Winkel psi_a und psi_b zurück.

CalcOptimalAngle() bestimmt den Optimalen Winkel ausgehend von der Geometrie des Moleküls: Unterscheidung, ob N oder O. Unterscheidung ob 1 Bindung oder 2 Bindungen. Die Funktion liefert die optimalen Winkel für phi und psi zurück. Vgl. Referenzcode Zeile 1063

CalcCosPhi() bekommt die Winkel phi und psi, die optimalen Winkel von phi und psi, und die relevanten Atome eines jeden Konstruktes. Es wird die Differenz zwischen Optimaler Winkel und berechneter Winkel gebildet. Zusätzlich wird der cosinus berechnet. Wenn es

nicht genügend Nachbaratome gibt für atom_1 (N,O) bzw. atom_2(N,O), so können die Winkel phi bzw. psi nicht korrekt ermittelt werden. Daher werden diese Werte auf 1.0 gesetzt. (Angelehnt an Referenzcode Zeile 1270). Die Funktion liefert letztlich cos_phi und cos_psi für atom_1 bzw. atom_2.

calcHBondEnergy() wird nur aufgerufen, wenn es sich um ein Konstrukt handelt, welches ein cos_theta > 0 aufweist. Die Funktion wird mit allen bisher kalkulierten Termen des Konstruktes aufgerufen, wobei die in CalcHbonds initialisierte Summe ebenfalls als Übergabeparameter übergeben wird, um die Summe aller Energien zu erhalten. Die Berechnung der Energie erfolgt mit Hilfe der Formeln aus dem Paper: „A third generation dispersion and a third-generation hydrogen bonding corrected PM6 method: PM6-D3H+". Die Parameter Covcut, Shortcut und Longcut wurden in A.U konvertiert. Die Funktion liefert die Energiewerte in A.U. zurück.

Literatur:

1) „A third generation dispersion and a third-generation hydrogen bonding corrected PM6 method: PM6-D3H+" (Kromman et al. 2014).

2)https://www.gdch.de/fileadmin/downloads/Netzwerk_und_Strukturen/Fachgruppen/Analytische_Chemie/chemkrist/jones_1.pdf

3) Referenzcode: JensenGroup, 2014b GitHub

2.2 D3-Dispersion-Korrektur

getCoordinationNumber() berechnet die Koordinationszahl für jedes Atom mittels folgenden Formel:

$$CN^A = \sum_{B \neq A}^{N_{at}} \frac{1}{1+e^{-k_1\left(\frac{k_2(R_{A,cov}+R_{B,cov})}{r_{AB}}-1\right)}}$$

wobei, k1 = 16, k2 = 4.0/3.0.

copy_rc6() speichert die Referenz C6-Werte für jedes Atom Paar mit entsprechende Bindungsmöglichkeiten in einer Matrix ab.
Matrix besteht aus 94(Anzahl der Elemente) × 94 × 5(Bindungsmöglichkeiten) × 5 Einträge:

[[[Referenz c6-Wert], [CN_A], [CN_B]], [[], [], []], [[], [], []], [[], [], []], [[], [], []]]

0-Bindung 1-Bindung 2-Bindunge 4-Bindungen

getdispersionscofficient berechnet Dispersionskoeffizient c6 für jedes Atom Paar mit folgenden Formel:

$$C_6^{AB}(CN^A, CN^B) = \frac{Z}{W}$$
$$Z = \sum_i^{N_A} \sum_j^{N_B} C_{6ref}^{AB}(CN_i^A, CN_j^B) L_{ij}$$
$$W = \sum_i^{N_A} \sum_j^{N_B} L_{ij}$$
$$L_{ij} = e^{-k_3[(CN^A - CN_i^A)2 + (CN^B - CN_j^B)2]}$$

wobei, $L_{ij}: Gaussian - Distanz$,

k3 = -4.0

Z ergibt sich aus der Summe den Referenz c6-Koeffizienten über die Bindungsmöglichkeiten von Atom A und Atom B. W ist die Summe der Gaussian-Distanzen über die Bindungsmöglichkeiten und Atom A und Atom B.
Um die Dispersionskoeffizient c6 zu berechnen, wird zuerst alle Bindungsmöglichkeiten für jedes Atom in Matrix copy_c6 gesucht und in der Variable mxc[idx] gespeichert.
Die ausgewerteten Dispersionskoeffizienten für Atom Paar AA und AB werden in entsprechendem Array gespeichert und zurückgegeben.

calculateDispersionEnergy() berechnet die Dispersionsenergie mittels Formel:

$$E_{disp}^{D(CSO)} = -\sum_{AB} \left[s_6 + \frac{a_1}{1+\exp(R_{AB} - a_2 R_0^{AB})} \right] \times \frac{C_6^{AB}}{R_{AB}^6 + (a_3 + R_0^{AB} + a_4)6}$$

wobei, a1, a2, a3, a4, s6 sind Skalierungsfaktoren,

mit a1 = 1.0, a2 = 4.5, a3 = 0.0, a4 = 2.52, s6 = 1.0

R0AB: Kovalent Radius für Atom Paar AB.

$$R_0^{AB} = \sqrt{3 \cdot r2r4[A'] \cdot r2r4['B']}$$

Literatur:

Grimme,S., Antony,J, Ehrlich,S., Kneg,Helge. A consistent and accurate ab initio parametrization of density functional dispersion correction (DFT-D) for the 94 elements H-Pu. THE JOURNAL OF CHEMICAL PHYSICS 2010,132, 154104.

Schröder.H., Creon.A., Schwabe.T. Reformulation of the D3(Becke-Johnson) Dispersion Correction without Resorting to Higher than C_6Dispersion Coefficients. J.Chem.Theory Comput. 2015,11, 3163-3170.

Referenzcode: pyDFTD3, April, 2016 GitHub

https://github.com/bobbypaton/pyDFTD3

2.3 Elektrostatische Energie

calculateElectrostaticEnergy() berechnet die elektrostatische Energie E_{ES} eines Moleküls mit der Formel:

$$E_{ES} = \sum_{A} \sum_{B \neq A} \frac{q_A q_B}{R_{AB}}$$

wobei: q die Partialladung eines Atoms ist,
A und B Atomindizes sind und
R die Distanz zwischen zwei Atomen ist.

Die Berechnung von E_{ES} wird durch zwei ineinander geschachtelte for-Schleifen realisiert. Zudem sind die Werte der Partialladungen aller im Molekül enthaltenen Atome erforderlich. Die Kalkulation der Partialladungen erfolgt über die Funktion calculatePartialCharges.

calculatePartialCharges() berechnet die Partialladungen $[q_1 \ldots q_N]$ aller Atome im Molekül

$$q_A = \sum_{B \neq A} f_{damp}(A, B)(EN_B - EN_A) \cdot 0{,}08$$

und gibt diese als Vektor aus. Die Berechnung erfolgt mit der Formel:

wobei: q die Partialladung eines Atoms ist,
A und B Atomindizes sind,
EN die Elektronegativität eines Atoms,
$f_{damp}()$ eine Korrekturfunktion und
0.08 ein Korrekturwert ist.

Die Partialladungen sind damit abhängig von den Elektronegativitäten der Atome. Die Korrekturfunktion $f_{damp}()$, gibt einen Dämpfungswert wieder, der die Distanzen der Atompaare berücksichtigt, für die die Differenz der Elektronegativität gebildet wird. Der Wert 0,08 ist eine dimensionslose Zahl und ein empirischer ermittelter Parameter, der als Korrekturfaktor dienen soll.

Die so ermittelten Partialladungen werden anschließend korrigiert. Hierfür wird die CM5-Korrektur[1] angewendet, die in calculatePartialCharges() als Funktion cm5correction() eingebettet ist.

cm5correction() ist eine Korrekturfunktion. Diese wird angewendet, da das Model zur Berechnung der Partialladungen Dipolmomente unterschätzt, wodurch die Partialladungen schwächer ausfallen.

Die Formeln für die Bestimmung CM5-korrigierten Partialladungen lautet:

$$q_A^{CM5} = q_A + \sum_{B' \neq A} T_{AB'} B_{AB'}$$

wobei q_A^{CM5} die CM5-korrigierte Partialladung,
q_A die Ausgangspartialladung,
T ist ein empirisch ermittelter Koeffiziente,
A und B‘ Atomindizes sind,
B separat in der Funktion getProductTB() ermittelt wird.

getProcuctTB() wird in der CM5-Korrektur aufgerufen und das Produkt aus dem empirisch ermittelten Koeffizienten T und dem Wert B gebildet. B wird mittels der folgenden

$$B_{AB'} = \exp\left[-\alpha\left(r_{AB'} - R_{Z_A} - R_{Z_{B'}}\right)\right]$$

Gleichung berechnet:

wobei R der kovalente Radius,
r die Distanz zweier Atome ist,
α ist ein empirisch ermittelter Koeffizient und
A und B‘ Atomindizes sind.

Der Koeffizient T wird mit der Funktion get_T() ermittelt.

get_T() gibt den Koeffizienten T aus. T ist abhängig vom Atompaar AB, auf das es sich bezieht. Die Funktion extrahiert die Werte in Abhängigkeit zum Atompaar aus dem Dictionary DZ_params, welches in constants.py gespeichert ist. Die Berechnung von T erfolgt über:

$$T_{AB} = D_{Z_A} - D_{Z_B}$$

wobei D eine Konstante ist, die von
Z, der Atomzahl der jeweiligen Atome A und B ist.

Es werden hierbei werden spezielle Atomkombinationen berücksichtigt, deren Werte nicht in DZ_params codiert, sondern separat in det_T eingepflegt wurden.

f_damp() ist eine Funktion, die den Dämpfungsfaktor f_{damp} berechnet, der in die Berechnung der Partialladung mit einfließt[2]. Die Berechnung erfolgt mit der Formel:

$$f_{damp}(A,B) = \frac{1.0}{1.0 + e^{-k1\left(k2\left(\frac{R_{A,cov}+R_{B,cov}}{r_{AB}}\right)-1\right)}}$$

wobei: k_1 = 16 und k_2 = 4/3 Skalierungsfaktoren sind,
A und B Atomindizes sind,
R der kovalente Radius und
r die Distanz zweier Atome ist.

f_{damp} berücksichtigt bei der Kalkulation einer Partialladung die Atomdistanzen und gibt eine entsprechende Gewichtung wieder. Dadurch nehmen weit entfernte Atome nur geringfügig bis keinen Einfluss auf die Berechnung der Gesamtpartialladung.

def calcEnVector() liest alle als Vektor gespeicherten Atomtypen der Atome des Moleküls ein gibt deren Elektronegativitäten, die im Dictionary en_dict in der Datei constants.py gespeichert sind, als Vektor aus.

calcValenceVector() liest alle als Vektor gespeicherten Atomtypen der Atome des Moleküls ein gibt deren Valenzen, die im Dictionary valence_dict in der Datei constants.py gespeichert sind, als Vektor aus.

getCovVector_AA() gibt die Kovalenzradien der Atome im Molekül in der Einheit Ångstrom als Vektor aus. Die Funktion liest alle als Vektor gespeicherten Atomtypen der Atome des Moleküls und einen Vektor, welcher Elemente sortiert nach Ordnungszahl beinhaltet ein. Die Ordnungszahlen werden als Indizes der jeweiligen Atomtypen abgegriffen und dienen als Keys, um die entsprechenden kovalenten Radien aus dem Dictionary r_in_AA zu extrahieren. r_in_AA ist in der Datei cm5radii.py gespeichert.

getCovVector_AA() gibt die Kovalenzradien der Atome im Molekül in atomaren Einheiten als Vektor aus. Die Funktionsweise ist wie bei getCovVector_AA().

Literatur:

1. Marenich, A. V., Jerome, S. V., Cramer, C. J. & Truhlar, D. G. Charge model 5: An extension of hirshfeld population analysis for the accurate description of molecular interactions in gaseous and condensed phases. *J. Chem. Theory Comput.* **8,** 527–541 (2012).
2. Grimme, S., Antony, J., Ehrlich, S. & Krieg, H. A consistent and accurate ab initio parametrization of density functional dispersion correction (DFT-D) for the 94 elements H-Pu. *J. Chem. Phys.* **132,** (2010).

2.4 Born-Energie

Das Programm wurde geschrieben um die Lösungsmittelkorrektur für Moleküle mittels der folgenden Formel zu berechnen:

calculateFinal: nimmt die finale Berechnung der Born-Energie vor

$$\sum_{i<j}^{n} \frac{q_i q_j S_{GB}(r_{ij}^2 G_i G_j)}{r_{ij}} \sum_{i=1}^{n} \gamma_i (2R_i G_i)^3$$

$$\sum_{i=1}^{n} q_i^2 G^i -$$

$$E_{GB/VI} = \tau$$

$$\tau = (\varepsilon_1^{-1} - \varepsilon^{-1})$$

$$S_{GB}(t) = \frac{(1 + t^{-1} e^{-t}/4)^{-1}}{2}$$

Die Variablen:

ε1: dielektrische Konstante des "solute interior"

ε: dielektrische Konstante des LM

q: partielle Ladungen der Atome

G: "Born self energy factors"

rij: Distanz zwischen 2 Atomen

R: Radius des Atoms

γ: empirische Koeffizienten

dij : Referenz Bindungslänge zwischen 2 Atomen

S: Gelöste Moleküle werden als Set unabhängiger Sphären gesehen. Jedes Atom erhaelt einen S-Wert

calculateFormal(): Berechnung der „Born self energy factors“ erfolgt nach den nachstehenden Formeln:

$$-(2\mathrm{G}_i)^3 = R_i^{-3} - \sum_{j=1}^{n} V(r_{ij}, R_i, S_j)$$

$$V(r,R,S) = \begin{cases} L(r,x,S)\Big|_{x=max(R,r-S)}^{x=r+S} & |R-S| < r \\ 0 & 0 \le r \le R-S \\ L(r,x,S)\Big|_{x=r-S}^{x=r+S} + R^{-3} & 0 \le r \le S-R \end{cases}$$

$$L(r,x,S) = \frac{3}{2}\left[\frac{1}{4rx^2} - \frac{1}{3x^3} + \frac{r^2-S^2}{8rx^4}\right]$$

calculateSpheres(): Funktion zur Berechnung der Sphären-Werte nach folgender Formel:

$$S_i = 95max(0, v^{\frac{1}{3}})$$

$$v_i = R_i^3 - \frac{1}{8}\sum_j a_{ij}^2(3R_i - a_{ij}) + a_{ji}^2(3R_j - a_{ji})$$

$$a_{ij} = \frac{R_j^2 - (R_i - d_{ij})^2}{2d_{ij}}$$

zusätzliche Information: bei der Berechnung können verschiedene Werte für die Atomradien und die empirischen Koeffizienten verwendet werden. Dazu ist die supplement.py Datei zu editieren. Hierzu ist anzumerken, dass die Verwendung anderer Radien(aus anderen Quellen) keinen großen Einfluss auf den berechneten Wert haben wird. Dies stimmt mit den Aussagen der wissenschaftlichen Arbeit überein[1]. Die empirischen Koeffizienten haben jedoch einen großen Einfluss auf die Berechnung, sowie insgesamt der Rechenschritt bei dem diese verwendet werden. Für tieferes Verständnis der richtigen Verwendung der empirischen Koeffizienten verweise ich auf die wissenschaftliche Arbeit[1]. Das von mir erstellte Programm kann jedoch nur die beiden Ladungsmodelle AM1-BCC und MMFF94 verwenden.

Literatur:
[1]: P.Labute: The Generalized Born/Volume Integral Implicit Solvent Model: Estimation of the Free Energy of Hydration Using London Dispersion Instead of Atomic Surface Area, J Comput Chem 29: 1693–1698, 2008

1.1 Polarisierungsenergie

calculatePolarisationEnergy(atomPositions, numberOfAtoms, pointCharge, dispersionCoefficient, exclusionLists) berechnet die Polarisierungsenergie für ein gegebenes Molekühl.

Parameter:

- atomPositions: Liste, welche die Atompositionen beinhaltet.
  - Typ: Matrix ('Anzahl der Atome' x 3) vom Basistyp: Fließkommazahl.
- numberOfAtoms: Anzahl der Atome in der Datei. Typ: ganzzahlig.
- pointCharge: Vektor, welcher die Punktladungen der gegebenen Atome beinhaltet.
  - Typ: Vektor der Länge Anzahl der Atome' vom Basistyp: Fließkommazahl.

- dispersionCoefficient: Vektor, welcher die Dispersionscoeffizienten der gegebenen Atome beinhaltet.
  - Typ: Vektor der Länge Anzahl der Atome' vom Basistyp: Fließkommazahl.
- exclusionLists: Beinhaltet die Id des zugehörigen Molelühls für jedes Atom.
  - Typ: Vektor der Länge Anzahl der Atome' vom Basistyp: ganzzahlig.

Rückgabewert:
Polarisierungsenergie
Typ: Fließkommazahl

Die Berechnung erfolgt in mittels folgender Formel:

$$E = \frac{-1}{2}(\mu)^T F$$

Dabei ist μ ein Vektor, welcher sich folgendermaßen berechnet:

$$\mu = (\alpha^{-1} - T_2)^{-1} F$$

α ergibt sich durch die Dispersionskoeffizienten. Dabei wird in der Diagonale die Wurzel aus diesen gezogen.

$$\alpha = \begin{pmatrix} \alpha_{11} & 0 & \cdots & 0 \\ 0 & \alpha_{22} & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \alpha_{nn} \end{pmatrix}$$

$$F = -\sum T_\gamma^{ss\prime} q^{s\prime}$$

In diesem Programm werden ausschließlich statische Energien genutzt, für eine genauere zu berechnen müssten auch beispielsweise Quadrupol- oder Octupolmomente berücksichtigt werden.

Der Tensor zweiter Ordnung berechnet sich mit:

$$T_2 = \begin{pmatrix} 0 & \cdots & T_2^{1A} \\ \vdots & \ddots & \vdots \\ T_2^{A1} & \cdots & 0 \end{pmatrix}$$

Dabei ist jeder Eintrag in der Matrix eine 3x3 Matrix:

$$T_2^{ss\prime} = \begin{pmatrix} \Delta x \Delta x f & \Delta x \Delta y f & \Delta x \Delta z f \\ \Delta y \Delta x f & \Delta y \Delta y f & \Delta y \Delta z f \\ \Delta z \Delta x f & \Delta z \Delta y f & \Delta z \Delta z f \end{pmatrix}$$

mit: $f = \frac{1}{\sqrt{(x-x\prime)^2+(y-y\prime)^2+(z-z\prime)^2}}$

Literatur

- (klassische) Polarisationsenergie: http://dx.doi.org/10.1063/1.3560034

- Polarisationsparameter: http://dx.doi.org/10.1002/jcc.24425

Laufzeit
Durch die Matrixinvertierung liegt die Laufzeit in O(n^3).

Im folgenden werden Funktionen beschrieben, welche Teilberechnungen für die Hauptfunktion erfüllen:

_tensofunction_0(vector1, vector2) berechnet die Funktion f.
Parameter:

- vector1: Atomposition. Vektor der Länge 3.
- vector2: Atomposition. Vektor der Länge 3.

Rückgabewert: Typ: Fließkommazahl

Die Funktionen _tensofunction_1x(vector1, vector2), _tensofunction_1y(vector1, vector2) und _tensofunction_1z(vector1, vector2) berechnen die erste Ableitung der Funktion f nach x, y und z für zwei gegebene Positionen.
Parameter:

- vector1: Atomposition. Vektor der Länge 3.
- vector2: Atomposition. Vektor der Länge 3.

Rückgabewert: Typ: Fließkommazahl

Die zweite Ableitung der Funktion f ist ebenfalls ein Bruch, welcher sich in Zähler und Nenner aufteilt.
Die Funktionen
_calculate_tensor2xx_numerator(vector1, vector2),
_calculate_tensor2xy_numerator(vector1, vector2),
_calculate_tensor2xz_numerator(vector1, vector2),
_calculate_tensor2yy_numerator(vector1, vector2),
_calculate_tensor2yz_numerator(vector1, vector2)
und _calculate_tensor2zz_numerator(vector1, vector2) berechnen den Zähler der Funktion für die zweifachen entsprechenden Ableitungen nach x, y und z.

Ableitung nach xx: $2*(x-x')^2 - (y-y')^2 - (z-z')^2$
Ableitung nach xy = yx: $3.0*(x-x')*(y-y')$
Ableitung nach xz = zx: $3.0*(x-x')*(z-z')$
Ableitung nach yy: $-(x-x')^2 + 2*(y-y')^2 - (z-z')^2$
Ableitung nach yz = zy: $3.0*(y-y')*(z-z')$
Ableitung nach zz: $-(x-x')^2 - (y-y')^2 + 2*(z-z')^2$
Parameter:

- vector1: Atomposition. Vektor der Länge 3. Basistyp: Fließkommazahl.
- vector2: Atomposition. Vektor der Länge 3. Basistyp: Fließkommazahl.

Rückgabewert: Typ: Fließkommazahl

_calculate_tensor2_denominator(vector1, vector2) berechnen den Zähler der Funktion f für die zweifachen entsprechenden Ableitungen nach x, y und z. Da dieser für alle 9 Partialableitungen gleich ist, wird er nur einmal berechnet.
Parameter:

- vector1: Atomposition. Vektor der Länge 3. Basistyp: Fließkommazahl.
- vector2: Atomposition. Vektor der Länge 3. Basistyp: Fließkommazahl.

Rückgabewert: Typ: Fließkommazahl

_calculate_tensor_second_order(t2, p1, p2, vector1, vector2) füllt ein 3x3 Feld der Tensormatrix an der Stelle (p1,p2) mit denen aus vector1 und vector2 errechneten Werten für einen Tensor zweiter Ordnung.

Parameter:

- t2: Matrix, welche die Interaktionen zweiter Ordnung zwischen den Molekühlen repräsentiert.
  - Typ: Matrix der Größe (3 * 'Anzahl der Atome' x 3 * 'Anzahl der Atome')
- p1: x-Koordinate der Matrixposition.
  - Typ: ganzzahlig
- p2: y-Koordinate der Matrixposition.
  - Typ: ganzzahlig
- vector1: Atomposition. Vektor der Länge 3. Basistyp: Fließkommazahl.
- vector2: Atomposition. Vektor der Länge 3. Basistyp: Fließkommazahl.

_calc_f(atomPositions, pointCharge, ex) berechnet den zuvor beschriebenen Vektor F. Die selbe Position in jeder der drei gegebenen Listen muss dem selben Atom entsprechen.

Parameter:
- atomPositions: Liste mit Atompositionen.
  - Typ: Matrix der Größe 'Anzahl der Atome' x 3 mit Basistyp Fließkommazahl
- pointCharge: Entsprechende Punktladungen für jedes Atom in atomPositions.
  - Typ: Matrix der Größe 'Anzahl der Atome' x 3 mit Basistyp Fließkommazahl
- ex: Beinhaltet die Id des zugehörigen Molelühls für jedes Atom.
  - Typ: Vektor der Länge Anzahl der Atome' vom Basistyp: ganzzahlig.

Rückgabewert:
F: Vektor der Länge 3 * 'Anzahl der Atome' mit einer Fließkommazahl als Basistyp

_calc_alpha_inv(dispersionCoefficients) berechnet α^{-1} aus den Dispersionscoeffizienten.
Parameter:

- dispersionCoefficients: Liste mit Dispersionscoeffizienten.
  - Typ: Vektor der Länge 'Anzahl der Atome' mit einer Fließkommazahl als Basistyp

Rückgabewert:
α^{-1}
Typ: Matrix der Größe (3 * 'Anzahl der Atome' x 3 * 'Anzahl der Atome') mit Fließkommazahl als Basistyp.

_calc_t2(positions, exclusionList) berechnet die Interaktionen zweiter Ordnung zwischen den Atomen.
Parameter:

- positions: Liste mit Atompositionen.
  - Typ: Matrix der Größe 'Anzahl der Atome' x 3
- exclusionList: Beinhaltet die Id des zugehörigen Molelühls für jedes Atom.
  - Typ: Vektor der Länge Anzahl der Atome' vom Basistyp: ganzzahlig.

Rückgabewert:

- Matrix, welche die Interaktionen zweiter Ordnung zwischen den Molekühlen repräsentiert.
  - Typ: Matrix der Größe (3 * 'Anzahl der Atome' x 3 * 'Anzahl der Atome') mit Fließkommazahl als Basistyp.

_invert(matrix) invertiert die gegebene Matrix.
Parameter:

- matrix: quadratische 2-dimensionale Matrix mit numerischem Basistyp.

Rückgabewert:
invertierte Matrix

_calc_mu_ind(alpha_inv, t2, F) berechnet den zuvor erwänten Vector mu_ind nach folgender Formel mu_ind = (alpha^-1- t2)^-1 * F.
Parameter:

- alpha: Typ: Matrix der Größe (3 * 'Anzahl der Atome' x 3 * 'Anzahl der Atome')
- t2: Typ: Matrix der Größe (3 * 'Anzahl der Atome' x 3 * 'Anzahl der Atome')
- F: Vektor der Länge 3 * 'Anzahl der Atome' mit einer Fließkommazahl als Basistyp

Rückgabewert:
mu_ind: Vektor der Länge 3 * 'Anzahl der Atome' mit einer Fließkommazahl als Basistyp

_calc_Epol(mu_ind, F) berechnet die Polarisierungsenergie als
$E_{pol} = -1/2 * (mu_{ind})^T * F$.

Parameter:

- mu_ind: Vektor der Länge 3 * 'Anzahl der Atome' mit einer Fließkommazahl als Basistyp
- F: Vektor der Länge 3 * 'Anzahl der Atome' mit einer Fließkommazahl als Basistyp

Rückgabewert: Polarisierungsenergie als Fließkommazahl

3 Performanz

Die Performanz des Programms wurde 66 Molekülen getestet. Die entsprechenden Bindungsenergien die mit emmi berechnet wurden, wurden mit Referenzwerten aus einer anderen Implementation verglichen.

Unter Einbezug aller Energiefunktionen erzielte emmi einen Korrelationskoeffizienten von R = 0,251 mit einem Bestimmtheitsmaß von R^2 = 0,63 bei Vergleich mit Referenzenergien (Abbildung 2).

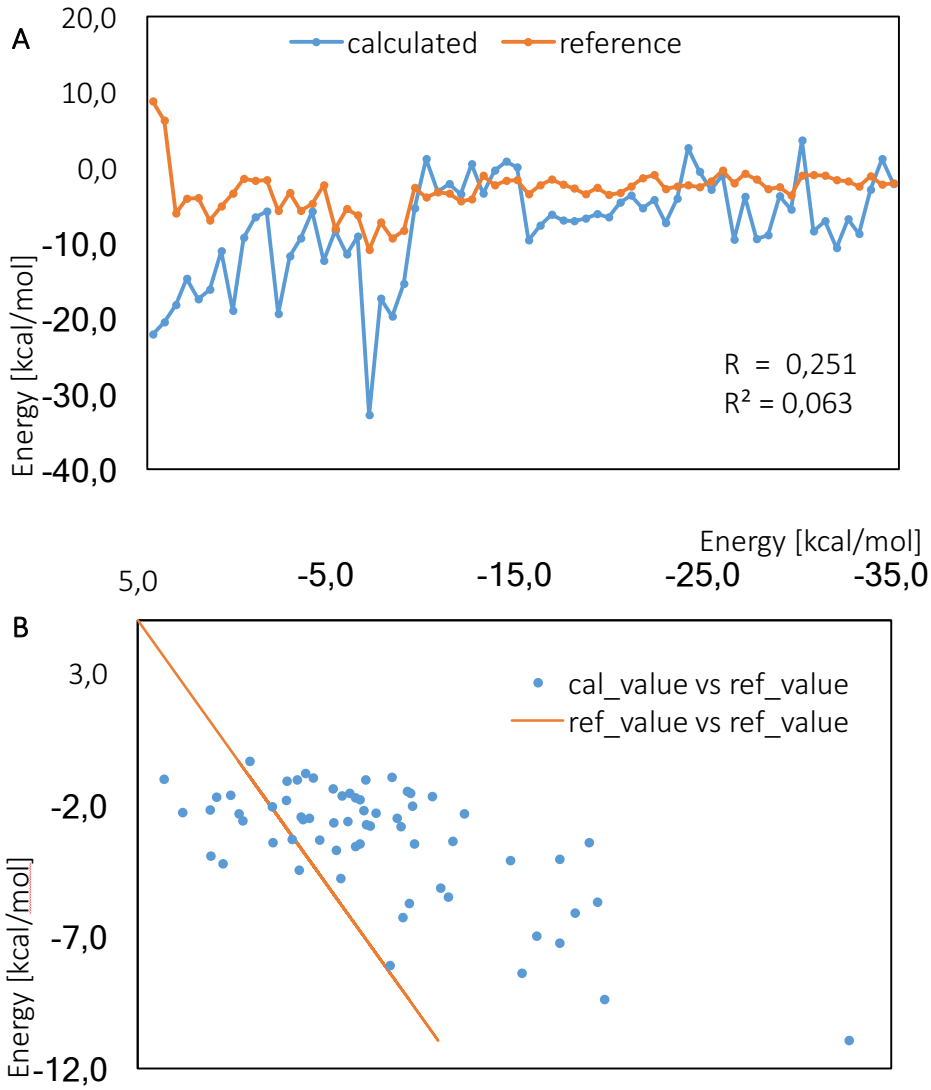

Abbildung 2: (A) Bindungsenergie mit allen Energien. (B) Korrelationsplot mit allen Energietermen.

Bei Weglassen des Energiebetrages der Lösungsmittelkorrektur betragen der

Korrelationskoeffizent R = 0,920 und das Bestimtmheismaß R^2 = 0,847 (Abbildung 3). Es wird hierbei deutlich, dass die Implementation für Born-Energie einen deutlichen Einfluss auf die Stärke der Korrelation mit den Referenzwerten hat. Unter Verwendung der Lösungsmittelkorrektur ist die Korrelation wesentlich geringer.

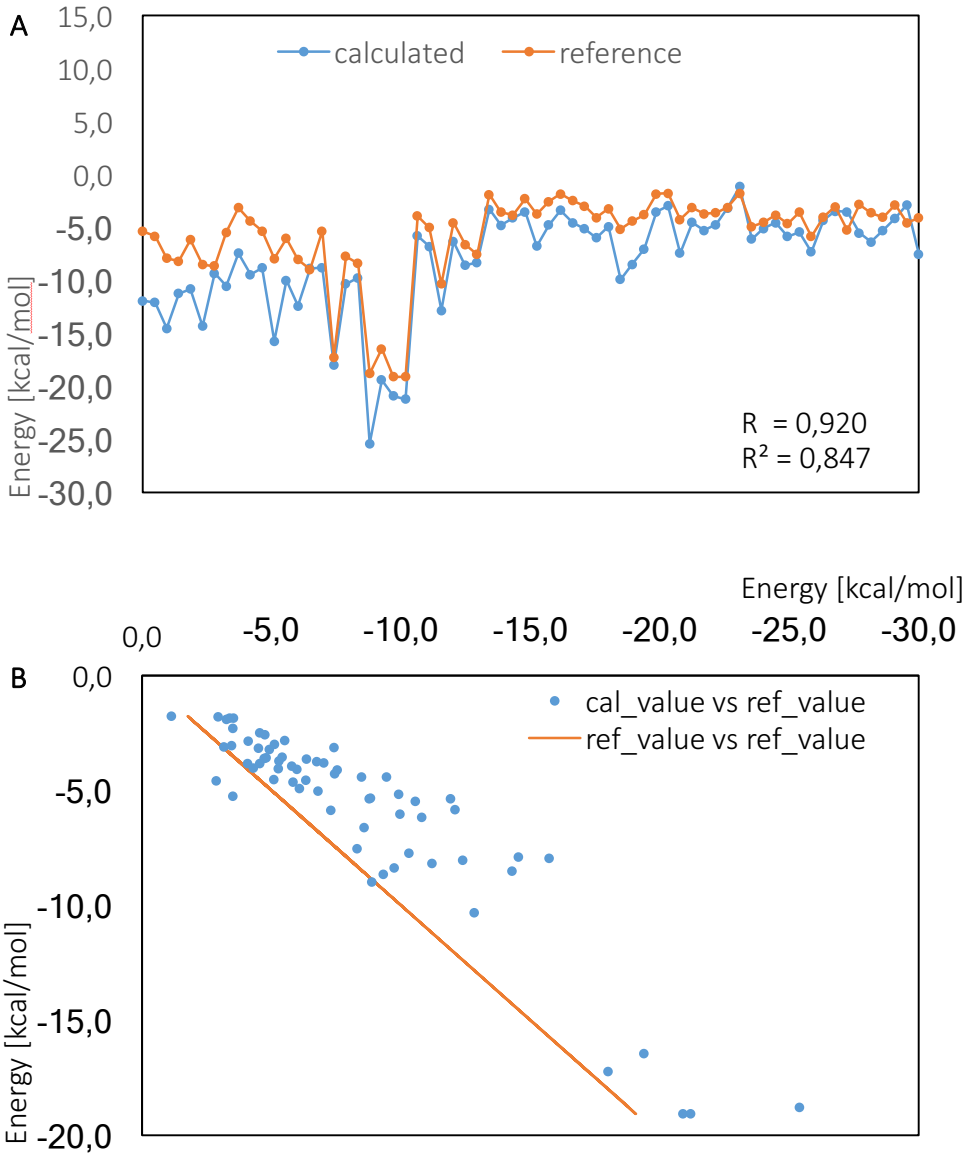

Abbildung 3: (A) Bindungsenergie ohne Lösungsmittelkorrektur. (B) Korrelationsplot ohne Lösungsmittelkorrektur.