Mini-MD in python

Ziel dieser Aufgabe ist es, selbst ein kleines MD-Programm zu schreiben, das zunächst nur Schwingungen zwischen zwei Atomen simuliert.
Erstellen Sie die Datei *mini-md.py.* Wie im Normalmodenteil mit MMTK arbeiten wir auch hier wieder in der Programmiersprache *python.*
Die Sprache zeichnet sich durch hohe Lesbarkeit aus, da Einrückungen (tabs) und Zeilenumbrüche zur korrekten Syntax dazugehören. Eine Befehlsdefinition wird immer durch einen Zeilenumbruch abgeschlossen, nicht wie z.B. in C, C++ oder PHP durch ein Semikolon. Was innerhalb einer Bedingung ausgeführt wird (z.B. *if, while* oder *for*), muss nicht wie bei anderen Sprachen in geschweifte Klammern gepackt werden, sondern wird einfach nur eingerückt. Dadurch ist auch die Fehlersuche vereinfacht, da diese in anderen Sprachen oft durch fehlende Klammern/Semikola verursacht werden.

1 Programmstruktur

Schreiben Sie in die erste Zeile eine Info an den Interpreter:

#!/usr/bin/python

Damit können Sie die Datei direkt ausführen, ohne *python* selbst aufzurufen.

Kommentare

Kommentare sind Teile des Programms, die vom Interpreter ignoriert werden und dazu dienen, sich selbst und anderen die Verständlichkeit des Codes zu erleichtern. In *python* ist alles, was in einer Zeile nach einem # steht, Kommentar. (Ausnahme: #!, siehe oben.)

Variablen

Variablen haben verschiedene Typen. Kommazahlen oder Floating point numbers (*float),* Ganzzahlen oder Integers (*int*), oder beliebige Zeichenfolgen oder Strings (*string*). Variablennamen dürfen nicht mit einer Zahl beginnen und aus Buchstaben, Zahlen, und Unterstrichen bestehen.

Beispiele für Variablendefinition:

- atom_id = 67
- b2 = 34.09
- VALUE = 67.9e8 # (= $67.9 \cdot 10^8$)
- object_5 = „Haus“ # strings mit Anführungsstrichen

Definieren Sie jetzt einige Variablen:

- u: atomare Masseneinheit
- nu: Schwingungsfrequenz von H-Cl (im Internet nachschauen), in 1/cm
- dt: MD-Zeitschritt in Sekunden, 1fs

Rechnen

Es gibt die vier Rechenoperatoren:

- * / mal, geteilt durch, z.B. x = 5 * y
- +, - plus, minus, z.B. index = index_1 + 8

Außerdem benutzen wir in diesem Programm

- pow(x, n) # x^n, Exponentialoperationen, z.B. square = pow(side_length, 2)
- abs(y) # |y| , Absolutwert, z.B. distance = abs(x-y)

Definieren Sie jetzt noch die Massen von Wasserstoff *m_H* und Chlor *m_Cl* in Einheiten von u.

Ausgabe

Kommandozeile

Die Ausgabe von Werten auf die Kommandozeile erfolgt über *print.*

Um die Ausgabe gut lesbar zu gestalten, gibt es Formatierungsplatzhalter. Sie bestimmen den Typ, in dem die Variable ausgegeben werden soll und die Anzahl der Stellen vor und nach dem Komma, die dafür freigehalten werden. Vergleichen Sie mögliche Ausgaben von m_H (machen Sie dazu ihr Programm ausführbar und starten Sie mit *./mini-md.py*):

- print „%f“ % m_H # Ausgabe als Kommazahl, standardmäßig mit 6 Nachkommstellen
- print „Die Masse von H ist %6.3f“ % m_H # Insgesamt 6 Zeichen, davon 3 hinter dem Komma
- print „Mass is %.30f“ % m_H # Float mit 30 Nachkommastellen
- print „Mass H: %e“ % m_H # Darstellung als Exponentialfunktion (siehe oben Variable VALUE)

oder von m_H und m_Cl

- print „%e\t%e“ % (m_H, m_Cl)

Welche Art der Ausgabe bevorzugen Sie?

In eine Datei

Um in eine Datei zu schreiben, erstellen Sie ein Dateizugriffsobjekt, ein Handle:

- md = open(„md.xyz“, „w“) # (falls nicht existent, erstelle, sonst) öffne die Datei „md.xyz“ zum Schreiben („w“ für write)

Testen Sie das Schreiben z.B. so:

- md.write(„Text zu Testen, und hier eine Variable %e\n“ % u)

Prinzipiell funktioniert die Textformatierung genauso wie bei *print.* Allerdings müssen Sie jede Zeile selbst mit einem Zeilenumbruch \n abschließen.

Ganz am Ende Ihres Programms (nicht nach jedem Schreibvorgang) schließen Sie die Datei, damit der Speicher freigegeben wird:

- md.close()

Sehen Sie sich „md.xyz“ an.

for-Schleife

In einer for-Schleife wird der Anfangswert einer Variablen so lange um einen bestimmten Wert hochgezählt, bis ein Abbruchkriterium erreicht wird. Innerhalb der Schleife wird jedesmal die gleiche Operation (oder Abfolge von Operationen) mit der erhöhten Variable durchgeführt.

Eine einfache for-Schleife in python sieht so aus:

```
for i in range(0,20):
    print „%d“ % i # %d ist Ausgabe als „digit“ also Ganzzahl ohne Komma
```

Das bedeutet: starte mit i=0 und erhöhe um 1, solange i<20. Gib in jedem Durchgang den Wert von i aus. Testen Sie diese Schleife in Ihrem Programm.

Diese Grundlagen sind ausreichend, um Ihr eigenes kleines MD-Programm zu schreiben und werden nun weiter angewendet.

2 Kraftfeld

Welche drei Potentialterme benötigen wir, um ein H-Cl-Molekül zu simulieren?

Wir beschränken uns für diese Aufgabe auf das harmonische Potential, das durch eine Kraftkonstante k und einen Gleichgewichtsabstand r0 definiert wird.

Berechnen Sie mit Hilfe Ihres Programms die Kraftkonstante aus der Schwingungsfrequenz und den Atommassen und definieren Sie damit die Variable k in Einheiten von kJ/(mol nm^2) . In dieser Einheit wird die Kraftkonstante auch in GROMACS angegeben. Überprüfen Sie in einer *ffbonded.itp*, ob die Größenordnung von k stimmt.

Definieren Sie für den Gleichgewichtsabstand die Variable r0 mit 0.1289 nm.

Sie sollten nun alle nötigen Parameter in Ihrem Programm definiert haben: m_H, m_Cl, k und r0.

3 Berechnung der Trajektorie

Zur Berechnung der einzelnen Zeitschritte nehmen wir den Verlet-Algorithmus (Vorlesung Kapitel 3C).

Anfangsbedinungen

Für eine Schwingung zwischen zwei Atomen reicht eine Dimension, wir nehmen x.

Definieren Sie zunächst für beide Atome Startpositionen xH_0 und xCl_0, in nm (entsprechend $x(t_0)$). Setzen Sie die Atome 2 Angström voneinander entfernt.

Definieren Sie auch die Anfangsgeschwindigkeiten vH_0 und vCl_0, jeweils als 0 (entsprechend $v(t_0)$).

Definieren Sie gemäß III.23 im Vorlesungsskript xH_old und xCl_old (entsprechend $x(t_0-\Delta t)$).

Damit Sie die Trajektorie später in VMD ansehen können, schreiben wir alle Zeitschritte der Trajektorie in eine Datei, und zwar im xyz-Format. Das Format sieht so aus:

2 # Anzahl der Atome

1fs # Kommentar, z.B. Zeitschritt

H x y z # Atomname und x, y, z Koordinaten in Angström

Cl x y z # Atomname und x, y, z Koordinaten in Angström

Schreiben Sie jetzt den ersten Zeitschritt $t_0=0$ in die Datei *md.xyz*.

Trajektorie

Öffnen Sie jetzt eine *for*-Schleife mit 20 Schritten.

- Berechnen Sie innerhalb der Schleife die Kräfte F_H und F_Cl auf die beiden Atome. Schreiben Sie sich die Gleichungen vorher sauber auf ein Stück Papier.

- Berechnen Sie daraus die jeweiligen Beschleunigungen a_H und a_Cl. Achten Sie auf die Einheiten. Da die Koordinaten in nm sind, muss die Beschleunigung in nm/s^2 sein.
- Benutzen Sie den Verlet-Algorithmus, um die neuen Positionen x_H und x_Cl (entsprechend $x(t+\Delta t)$ zu berechnen
- Schreiben Sie die Koordinaten jedes Schritts in *md.xyz*, gemäß des oben angegebenen Formats

Visualisierung

Wenn das Format stimmt, können Sie die Trajektorie in VMD ansehen. Welche Schwingungsdauer können Sie daraus ablesen? Stimmt diese mit der oben definierten Schwingungsfrequenz überein?

4 Dipol im externen Feld

Wir fügen nun ein festes elektrostatisches Potential hinzu, unter dessen Einfluss sich der H-Cl-Dipol bewegen soll. Um eine Drehung des Dipols zu ermöglichen, brauchen wir eine zusätzliche Dimension y. Wir gehen daher in die Vektorschreibweise über.

Kopieren Sie ihr bisheriges Skript unter einen neuen Namen. Definieren Sie zusätzliche Konstanten: ε_0 in $[C^2\,mol / kJ\,nm]$ und π.

Für die Vektoren brauchen wir *Numerical Python*, ein Paket, das viele wissenschaftlich notwendige Funktionen enthält. Importieren Sie numpy oben in Ihrem Skript (nach der Anweisung an den Interpreter, aber vor dem Programmcode):

```
import numpy as np
```

Durch diese Schreibweise können Sie auf Numpy-Funktionen mit der Abkürzung *np* zugreifen.

Vektoren

Ein n-dimensionaler Vektor wird mit np.array([]) definiert, z.B.

- A = np.array([1.0, 2.0, 3.0]) # Vektor mit x=1, y=2, z=3

Sie können ganz normal mit den Vektoren rechnen, testen Sie z.B.

- print A+2
- print A*5
- c = np.array([5.2, -9.0, 0.8])
- print c+A

Schreiben Sie ihr Programm komplett in 2-dimensionaler Vektorschreibweise um, also z.B. die Definition von x1 und x2 als 2-dimensionale Vektoren mit Namen r1 und r2. Sie können auf die einzelnen Einträge so zugreifen:

- print A[0] # erster Eintrag, entspricht x-Komponente

- print A[1] # zweiter Eintrag, entspricht y-Komponente

Sie können auch direkt einen Eintrag definieren, z.B.

- A[0] = 100

Der Betrag des Vektors wird mit np.linalg.norm(A) berechnet.

Potential

Wir fügen das Potential in Form von 2 entgegengesetzt geladenen, fest positionierten DUMMY-Teilchen hinzu.

Definieren Sie dazu zwei Vektoren RM und RP in von Ihnen gewählten Abständen zum Dipol.

Definieren Sie die Elementarladung e und dann für diese Teilchen jeweils eine Ladung QM und QP in Einheiten von e.

Partialladungen

Berechnen Sie aus dem Dipolmoment und dem Gleichgewichtsabstand von H-Cl die beiden Partialladungen q_H und q_Cl und definieren Sie die Variablen in Einheiten von e.

Trajektorie

Definieren Sie in Ihrer for-Schleife zunächst Vektoren für die Kräfte auf H und Cl:

- F_H = np.array([0.0, 0.0])
- F_Cl = np.array([0.0, 0.0])

Berechnen Sie die Kräfte auf H und Cl für jede Dimension. Schreiben Sie auch hier zunächst die Gleichungen sauber auf ein Blatt Papier.

Berechnen Sie daraus die Beschleunigungsvektoren a_H und a_Cl, und damit die neuen Positionen r1_new und r2_new, analog wie oben.

Schreiben Sie die Koordinaten zusammen mit denen der DUMMY-Atome in eine neue Datei *md_ESP.xyz* heraus und betrachten Sie die Trajektorie in VMD. Ist sie physikalisch sinnvoll?

5 2 Dipole im externen Feld

Erweitern Sie ihr Programm um einen weiteren H-Cl-Dipol.