**Übungsblatt 1:**
Quantenmechanischer Harmonischer Oszillator

Sommersemester 2016

# Programmiertechniken

Prof. Dr. L. Pollet

J. Greitemann, D. Hügel, J. Nespolo, T. Pfeffer

---

**1) Matrix Initialisierung und Diagonalisierung** Wir befassen uns mit dem quantenmechanischen harmonischen Oszillator mit Hamiltonian:

$$\hat{H} = \frac{\hat{p}^2}{2m} + \frac{1}{2}m\omega^2\hat{x}^2.$$

Der Impulsoperator $\hat{p}$ kann als $-i\hbar\frac{\partial}{\partial_x}$ umgeschrieben werden, sodass der Hamiltonian folgende Form annimmt:

$$\hat{H} = -\frac{\hbar^2}{2m}\frac{\partial^2}{\partial_x^2} + \frac{1}{2}m\omega^2\hat{x}^2.$$

Zur Vereinfachung nehmen wir nun als Masse:

$$m = \hbar^2$$

sowie als Frequenz:

$$\omega = \hbar^{-1}$$

sodass der Hamiltonian umgeschrieben werden kann als

$$\hat{H} = -\frac{1}{2}\frac{\partial^2}{\partial_x^2} + \frac{1}{2}\hat{x}^2.$$

Wir betrachten nun einen quantenmechanischen harmonischen Oszillator in einer eindimensionalen Box mit Länge $L$, wobei $x$ von $-L/2$ bis $L/2$ geht. Wir können also den Ort in $N$ Punkte

$$x_n = n\Delta_x - L/2$$

diskretisieren, wobei $\Delta_x = L/N$ ist und $n = 0, 1, ...N$. Genauso können wir nun die Wellenfunktion $\Psi$ im Ortsraum diskretisieren, als $\Psi_n = \Psi(x_n)$. Die partielle Ableitung in $x$ kann also umgeschrieben werden als:

$$\frac{\partial}{\partial_x}\Psi(x_n) = \frac{\Psi(x_n + \Delta_x/2) - \Psi(x_n - \Delta_x/2)}{\Delta_x}$$

und

$$\frac{\partial^2}{\partial_x^2}\Psi(x_n) = \frac{\Psi(x_n + \Delta_x) + \Psi(x_n - \Delta_x) - 2\Psi(x_n)}{\Delta_x^2} = \frac{\Psi_{n+1} + \Psi_{n-1} - 2\Psi_n}{\Delta_x^2}$$

sodass wir die zeitunabhängige Schrödingergleichung $\hat{H}\Psi = E\Psi$ im Ortsraum umschreiben können als

$$\hat{H}\Psi_n = -\frac{\Psi_{n+1} + \Psi_{n-1} - 2\Psi_n}{2\Delta_x^2} + \frac{1}{2}x_n^2\Psi_n$$

Der Hamiltonian besteht in dieser Basis also aus einer $N \times N$ Matrix mit folgenden Einträgen:

$$H_{n,n} = \frac{1}{2}x_n^2 + \frac{1}{\Delta_x^2}$$

$$H_{n\pm1,n} = -\frac{1}{2\Delta_x^2}$$

und allen anderen Einträgen gleich 0. Das Ziel dieser Aufgabe ist es, eine Klasse "HOSZ" zu schreiben, in der die Matrix $H$ initialisiert und diagonalisiert wird.

a) Wir schreiben eine Funktion "Matrix_HOSZ", welche die Anzahl an Ortspunkten $N$ und die Länge der Box $L$ als Argumente hat und daraus die $N \times N$ Matrix $H$ als numpy-array initialisiert. Das Array sollte so initialisiert werden, dass der Datentyp als 'int', 'float' und 'complex' eingestellt werden kann.

   1. Initialisieren Sie die Matrix für die drei verschiedenen Datentypen ('int', 'float' und 'complex' ) für $N = 100$ und $L = 10$. Nun kann die Matrix durch die Numpy-Funktion "linalg.eigh" diagonalisiert werden. Vergleichen Sie die Eigenwerte für die verschiedenen Datentypen.
   2. Vergleichen Sie die Laufdauer der Initialisierung und Diagonalisierung der Matrizen für die verschiedenen Datentypen. Dies kann durch das Python-Paket "time" gemacht werden: durch "t0 = time.clock()" am Anfang der Simulation wird die Anfangszeit der Simulation gespeichert; am Ende der Simulation kann durch "time.clock() - t0" die verstrichene Zeit ausgegeben werden.
   3. Welcher Datentyp ist optimal für dieses Problem?

b) Schreiben Sie eine Funktion "Eigensystem_HOSZ", die "Matrix_HOSZ" mit dem optimalen Datentyp verwendet um die Matrix zu initialisieren, diese dann mit "linalg.eigh" diagonalisiert und die Eigenwerte und -Vektoren ausgibt.

**2) Vergleich mit Analytischen Lösungen**

Die Analytische Lösung des Harmonischen Oszillators in einem unendlich großen System, $L \to \infty$, besteht aus Eigenfunktionen $\Psi^l$ mit Eigenwerten $E_l$, wobei

$$E_l = l + \frac{1}{2}$$

$$\Psi^l(x) = \frac{1}{\sqrt{2^l l!}}\pi^{-\frac{1}{4}}e^{-\frac{x^2}{2}}H_l(x)$$

mit den hermiteschen Polynomen

$$H_l(z) = (-1)^l e^{z^2}\frac{d^l}{dz^l}\left(e^{-z^2}\right)$$

a) Fügen Sie die Funktion "Analytische_Lsg_HOSZ" zur Klasse bei, welche die $l$-te Eigenenergie $E_l$, sowie die $l$-te Eigenfunktion $\Psi^l$ als Vektor im diskreten Ortsraum mit Einträgen

$$\Psi^l_n = \Psi^l(x_n)$$

ausgibt. Hierzu kann die scipy Funktion "special.hermite" verwendet werden, um die hermiteschen Polynome zu berechnen.

b) Fügen Sie eine Funktion "Vergleich_Lsg_HOSZ" hinzu, die mit "Eigensystem_HOSZ" und "Analytische_Lsg_HOSZ" sowohl die numerischen als auch die analytischen Eigenwerte und Eigenvektoren berechnet. Außerdem sollte die Funktion einen Plot ausgeben in dem die ersten 10 Eigenwerte aus beiden Methoden zum Vergleich geplottet werden, sowie 10 weitere plots in denen die Dichteverteilung $n^l(x)$ der ersten 10 Eigenvektoren $\Psi^l$ im Ortsraum zum Vergleich geplottet werden, wobei

$$n^l(x_n) = \left|\Psi^l_n\right|^2$$

c) Vergleichen Sie die Ergebnisse für $L = 10$ und $N = 100$. Führen Sie das selbe für $L = 1$ durch, was ändert sich?

d) Schreiben Sie eine weitere Funktion "Numerischer_Fehler_HOSZ", welche den numerischen Fehler der ersten 10 Eigenwerte ausgibt, der berechnet wird als

$$\Delta E = \sum_{l=0}^{9} |E_l - e_l|$$

wobei $E_l$ der $l$-te analytische Eigenwert und $e_l$ der $l$-te numerische Eigenwert ist. Plotten Sie den Fehler $\Delta_E$ für $L = 100$ und $N = 500, 600, 700, ..., 1500$ als Funktion von $\Delta_x = L/N$.

e) Fitten Sie den Fehler $\Delta_E(\Delta_x)$ mit einer Funktion

$$f(\Delta_x) = \sum_{n=0}^{4} a_n \Delta_x^n$$

durch die scipy funktion "scipy.optimize.curve_fit". Welche ist die Ordnung $m$ des Fehlers

$$\Delta_E \approx \mathcal{O}\left[\Delta_x^m\right]$$

Woran könnte das liegen?