Abschlussprojekt Wissenschaftliches Programmieren

Dozent: Bálint Aradi
http://www.bccms.uni-bremen.de/cms/people/b_aradi

SoSe 2020

Spätester Abgabetermin: 04.10.2020

1 Allgemeine Aufgabenstellung

Jeweils zwei Personen als eine Entwicklergruppe sollten ein wissenschaftliches Programmierprojekt verwirklichen. Für die Bewertung des Projektes muss das **Repository** abgegeben werden, das die Entwicklungsgeschichte enthält. Zusätzlich zur Begutachtung des Projektrepositories findet ein kurzes Auswertungsgespräch (ca. halbe Stunde) statt, in dem die Entwickler Fragen zu ihrem Projekt beantworten müssen. Unabhängig davon, welche Teile des Projektes von wem umgesetzt wurden, müssen alle Entwickler mit allen Teilen des Projektes vertraut sein.

Die Abgabe erfolgt, indem das Repository dem Dozenten als *tar.xz*-Archiv per Email zugeschickt wird. Die Email sollte auch Namen, Matrikelnummer und Email beider Entwickler enthalten. Die Abgabe wird durch den Dozenten bestätigt, in dem er eine von ihm digital signierte Version des Archives an beiden Entwickler zurückschickt. Gleichzeitig wird er einen Zeitpunkt für das Auswertungsgespräch vorschlagen.

2 Projektübersicht

Das Projekt muss bei der Abgabe folgende Komponenten enthalten:

- Ein ausführbares Python-Programm zur Lösung der eindimensionalen zeitunabhängigen Schrödingergleichung für beliebige Potentiale. Das Programm soll eine Inputdatei (`schrodinger.inp`) einlesen, in der die Details des 1D-Problems spezifiziert werden, die Schrödingergleichung lösen und die relevanten berechneten Größen in entsprechend benannten Dateien speichern.

- Ein ausführbares Python-Programm zur Visualisierung der Ergebnisse. Es soll die vom Gleichungslöser geschriebene Dateien einlesen und die Ergebnisse (Potential, Eigenwerte, Eigenfunktionen, Erwartungswerte) graphisch darstellen.

- Tests (unit-tests), die die Funktionalität des Gleichungslösers ausgiebig testen.
- Ausreichende Benutzerdokumentation (vorzugsweise in Englisch), damit andere Benutzer das Programmpaket eigenständig benutzen und die erhaltenen Ergebnisse interpretieren können.
- Ausreichende Programmierdokumentation (API-Dokumentation), damit die Funktionalität des Programmpaketes auch in anderen Programmierprojekten wiederverwendet werden kann.

3 Allgemeine Ausführungsvorgaben

Bitte beachten Sie bei der Realisierung des Programmierprojektes unbedingt folgende Vorgaben:

- Das Programmierprojekt muss in der Programmiersprache Python (Version 3) realisiert werden.
- Bei der Realisierung des Programmierprojektes dürfen nur die Standardmodule von Python und die externen Pakete numpy, scipy, matplotlib und pytest verwendet werden.
- Das Projekt muss sinnvoll modularisiert werden und mindestens zwei Module enthalten. Die Module müssen in ein Python-Paket eingegliedert sein, das alle Module des Projektes enthält.
- Alle Python Quellcodedateien müssen einen Pylint-Score von mindestens 9.0 haben. (Bei der Überprüfung sollte die vorgegebene Pylint-Konfigurationsdatei verwendet werden.)
- Die API-Dokumentation soll via Sphinx-extrahiert werden können. Die entsprechenden Konfigurationsdateien und ReST-Textdateien sollten als Teil des Projektes im Repository enthalten sein. (Sie sollten darauf achten, nur solche Objekte / Funktionen öffentlich zu machen, für die das tatsächlich sinnvoll ist. Machen Sie alle andere Objekte privat, um Implementationsdetails möglichst gut zu verstecken.)
- Ein Teil des Projektes sollte parallel (in zwei getrennten Repositories oder Zweige) entwickelt und dann zusammengeführt werden. (Das abgegebene Repository muss mindestens ein Merge enthalten!)
- Grundsätzlich gilt: Sie sollten demonstrieren, dass Sie die **im Kurs vermittelten Sprachelemente, Programmiertechniken und Entwicklungsworkflows verstanden und verinnerlicht** haben.

- Sie sollten auch **den wissenschaftlichen Teil des Projektes verstehen**. Überlegen Sie sich, welche Näherungen in die Berechnungen eingehen und welche Konsequenzen diese auf die Resultate haben. In den Fällen, wo das Problem auch analytisch lösbar ist, sollten Sie die numerischen Ergebnisse mit den analytischen vergleichen. Überlegen Sie, woher eventuelle Abweichungen stammen könnten. Wenn die Lösungen analytisch nicht ermittelt werden können, überlegen Sie, ob die erhaltenen Lösungen zumindest plausibel sind.

- Die Ein- und Ausgabedateien sollten *exakt* die Namen und die Formate haben, wie in der Spezifikation angegeben, damit die Ergebnisse Ihres Gleichungslösers auch mit nicht von Ihnen geschriebenen Tools überprüft werden können.

4 Detailierte Ausführungsangaben

4.1 Gleichungslöser

- Das Programm soll die eindimensionale zeitunabhängige Schrödingergleichung in einem frei wählbaren Bereich lösen.

- Die Schrödingergleichung soll in dem entsprechendem Bereich diskretisiert und mittels Diagonalisierung gelöst werden. (Siehe mathematische Details weiter unten.)

- Die Anzahl der zur Diskretisierung verwendeten Punkte wird vom Benutzer in der Inputdatei festgelegt. Ebenso legt der Benutzer die Masse des Teilchens und das externe Potential (V) fest.

- Das Potential wird durch Stützpunkte (beliebiger Anzahl) festgelegt. Der Benutzer darf frei auswählen, ob zwischen den Stützpunkten stückweise linear oder mit **natürlichem** kubischen Spline interpoliert wird oder (mit Berücksichtigung aller Stützpunkten) eine Polynominterpolation verwendet wird.

- Das aus den Stützpunkten erzeugte diskretisierte Potential soll (zusammen mit den Koordinaten der Diskretisierungspunkte) in der Datei `potential.dat` in XY-Format gespeichert werden, sodass es mit graphischen Tools dargestellt werden kann. Das XY-Format ist wie folgt:

  ```
  x1 V(x1)
  x2 V(x2)
  :
  ```

- Der Benutzer sollte frei auswählen können, welche Eigenwerte und Wellenfunktionen berechnet und gespeichert werden sollten (von bis). Die ausgewählten Eigenwerte sind

in der Datei `energies.dat` (ein Eigenwert pro Zeile) und die *normierten* Wellenfunktionen in `wavefuncs.dat` zu speichern. Letztere soll das NXY-Format haben, indem in jeder Zeile eine x-Koordinate und die dazugehörende Werte aller berücksichtigten Wellenfunktionen stehen:

```
x1 wf1(x1) wf2(x1) wf3(x1) ...
x2 wf1(x2) wf2(x2) wf3(x2) ...
:
```

- Zusätzlich zur Lösung des Eigenproblems, sollten der Erwartungswert des Ortsoperators $\langle\hat{x}\rangle$ und die erwartete Unschärfe der Ortsmessung $\sigma_x = \sqrt{\langle\hat{x}^2\rangle - \langle\hat{x}\rangle^2}$ für jeden Eigenzustand berechnet werden. Diese Erwartungswerte sollten in der Datei `expvalues.dat` gespeichert werden. Jede Zeile der Datei soll die entsprechenden 2 Werte für einen Eigenzustand enthalten.

- Die Benutzerangaben sollten aus der Datei `schrodinger.inp` eingelesen werden. Alle Größen in der Ein- und Ausgabe sollten in atomaren Einheiten angegeben sein.

- Es sollte mithilfe einer Kommandozeilenoption möglich sein, das Verzeichnis anzugeben (falls es nicht das aktuelle Verzeichnis ist), in dem die Inputdatei sich befindet und in das die Ergebnisse geschrieben werden.

- Der ausführbare Skript selber sollte nur möglichst wenig Code enthalten. Die Hauptfunktionalität soll in entsprechenden Modulen implementiert werden.

4.2 Visualisierer

- Das Programm soll die Dateien, die der Gleichungslöser erstellt hat, einlesen und graphisch darstellen (entweder direkt am Bildschirm anzeigen oder in eine PDF-Datei schreiben).

- Wenn die Dateien sich nicht im aktuellen Verzeichnis befinden, sollte es möglich sein, per Kommandozeilenparameter das Verzeichnis anzugeben.

- Sollten zur sinnvollen graphischen Darstellung noch weitere Parameter nötig sein (z.B. Faktor zur Skalierung der Wellenfunktionen zur besseren Darstellung, dargestellte X- und Y-Bereiche, usw.), so sollten diese als Kommandozeilenparameter an das Programm übergeben werden können. (Man sollte also sinvolle Abbildungen erzeugen können, ohne dass man den Quellcode des Visualisierers ändern muss.)

- Der Visualisierer sollte einerseits das Potential, die Eigenwerte und zu den Eigenwerten gehörende Wellenfunktionen darstellen, anderseits die für die Eigenwerte berechnete Erwartungswerte anzeigen. (Siehe Beispiele weiter unten.)

- Der ausführbare Skript selber sollte nur möglichst wenig Code enthalten. Die Hauptfunktionalität soll in entsprechenden Modulen implementiert werden.

4.3 Tests

- Es sollen automatische Tests als Teil des Projekts angelegt werden, die die Funktionalität des Lösers testen. Die Tests sollten automatisch ausgeführt werden, wenn man `python3 -m pytest` im obersten Verzeichnis des Projektes ausführt.
- Die Tests sollten vorgespeicherte Inputdateien mit unterschiedlichen Potentialen einlesen, die Schrödingergleichung lösen, und die Ergebnisse mit gespeicherten Referenzergebnissen vergleichen.
- Es sollten sowohl das diskretisierte Potential als auch die Eigenwerte auf Richtigkeit getestet werden. (Überlegen Sie sich, welche Probleme beim Testen von anderen Größen, wie z.B. die Eigenfunktionen auftreten könnten, und wie diese eventuell behoben werden könnnten. Sie müssen das aber nicht unbedingt implementieren.)
- Es sollten Tests für alle unten angegebene Beispiele erstellt werden.

4.3.1 Erstellung der Referenzdaten für die Tests

Die Referenzdaten für die Tests sollten folgenderweise erstellt werden:

1. Die vorgegebenen Fälle werden mit dem Programm berechnet.
2. Es wird **kritisch** überprüft, ob die erhaltenen Lösungen richtig sind. In den Fällen, bei denen die analystische Lösung bekannt ist, sollten die Ergebnisse mit diesen verglichen werden. In den anderen Fällen sollte es überlegt werden, ob die erhaltenen Lösungen zumindest physikalisch und mathematisch sinnvoll sind.
3. Wenn die Ergebnisse nach der kritischen Überprüfung als korrekt eingestuft werden, sollten diese als Referenz gespeichert werden.

4.4 Dokumentation

- Das Projekt sollte genug Benutzerdokumentation enthalten, damit ein Benutzer ohne weitere Hilfe die Programme sinnvoll benutzen und die Ergebnisse interpretieren kann.
- Die Dokumentation sollte entweder in der README-Datei stehen oder Teil der mit Sphinx-erstellten Dokumentation sein. (Im letzteren Fall sollte in der README-Datei darauf hingewiesen werden.)

- Zusätzlich zur Benutzerdokumentation sollten die öffentlichen Module / Funktionen entsprechend dokumentiert werden, sodass diese mit Sphinx extrahiert werden können (API-Dokumetation). Die Konfigurationsdateien und ReST-Quellcodedateien des entsprechenden Sphinx-Projektes sollten Teil des Projektes (des Projektrepositories) bilden.

4.5 Versionsgeschichte

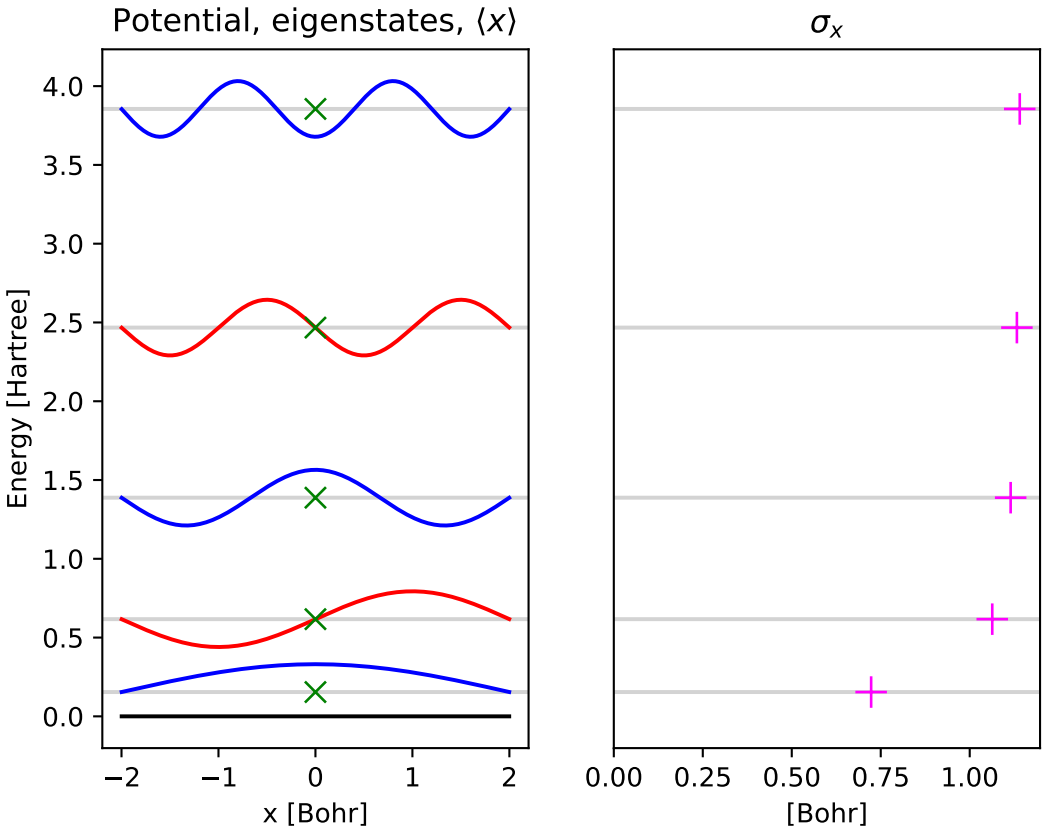
- Die verschiedenen Stufen der Programmentwicklung müssen mit dem Versionsverwaltungssystem Git festgehalten werden.
- Beide Entwickler müssen nach dem Projektstart eigene Repositories anlegen und ihre Veränderungen immer in das eigene Repository einchecken.
- Die Repositories müssen dann so oft wie erforderlich synchronisiert werden (merge). *Abzugeben ist nicht nur das fertige Programm, sondern auch eines der beiden git-Repositories*, das den Code in seinem Endzustand und mindestens 4-5 Revisionen (mit mindestens einem *merge*) enthalten muss.

5 Anwendungsbeispiele

Nachfolgend werden Anwendungsbeispiele angegeben, jeweils mit dem Inhalt der Eingabedatei `schrodinger.inp` und der graphischen Darstellung der Ergebnisse. *Das abgegebene Programm muss die aufgeführten Eingabedateien unverändert, so wie sie hier stehen, lesen können und aus den Ergebnissen müssen mit dem Visualisierer die gezeigten (oder ähnliche) Abbildungen erzeugt werden können.*

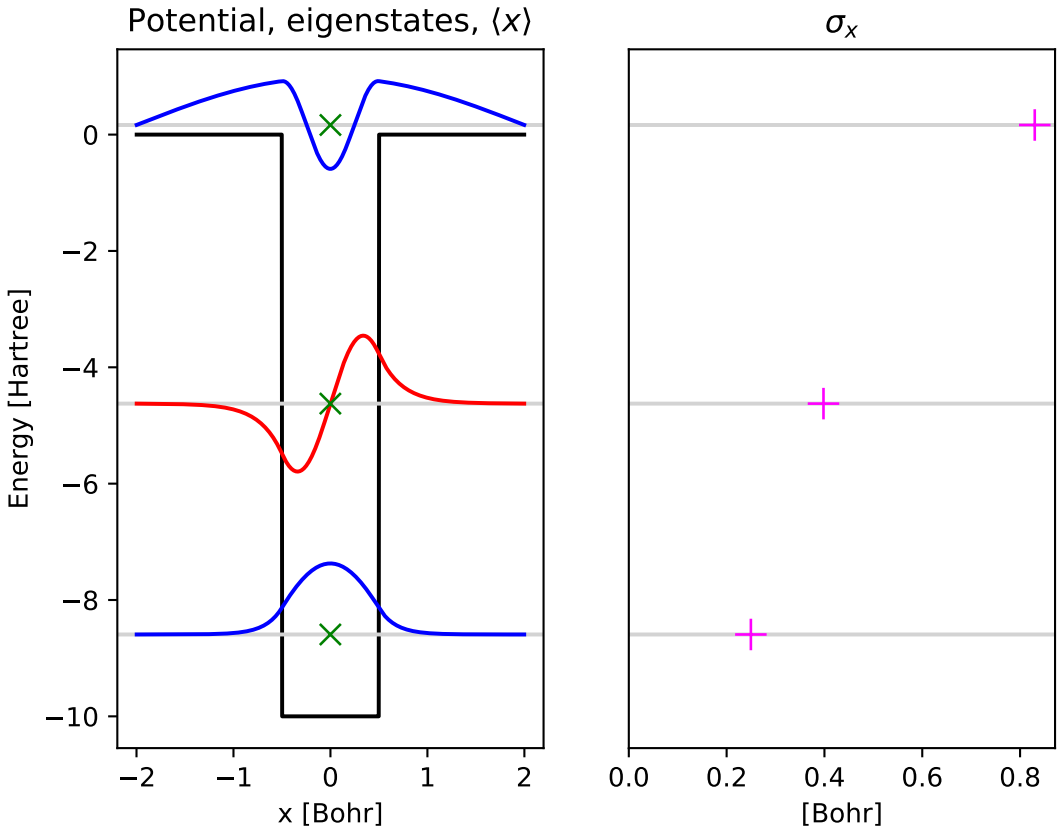
5.1 Unendlich tiefer Potentialtopf

```
2.0            # mass
-2.0 2.0 1999  # xMin xMax nPoint
1 5            # first and last eigenvalue to print
linear         # interpolation type
2              # nr. of interpolation points and xy declarations
-2.0  0.0
 2.0  0.0
```



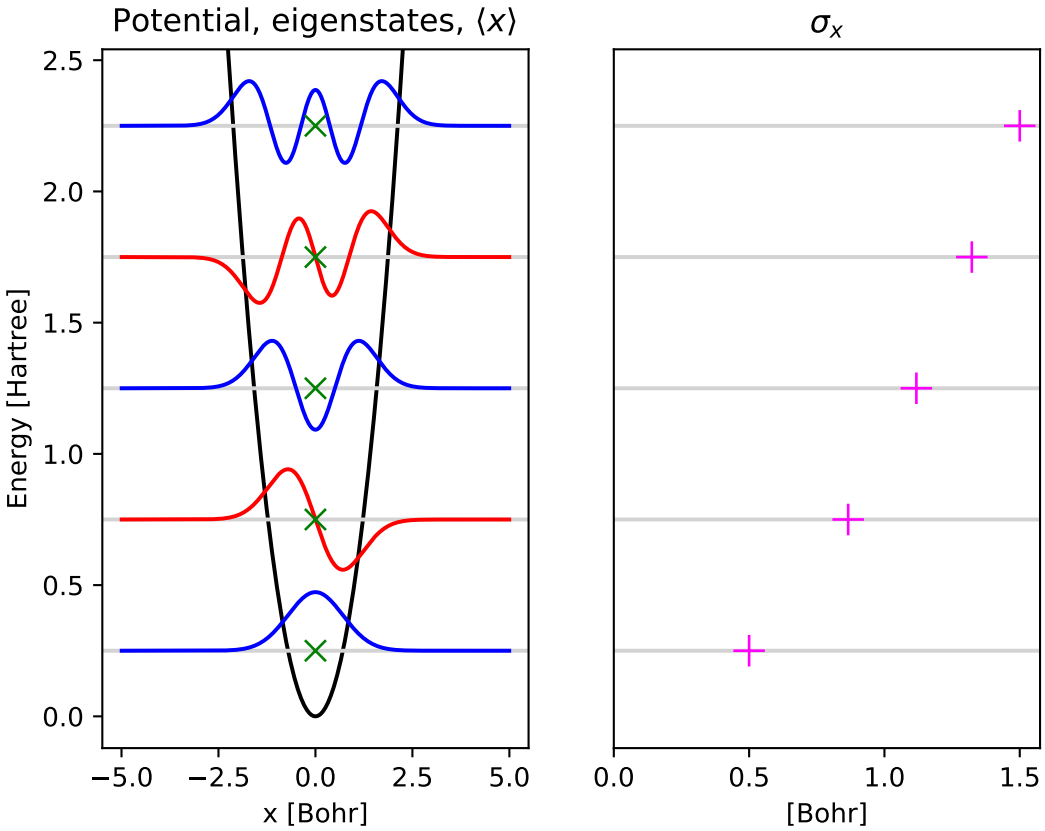
5.2 Endlich tiefer Potentialtopf

```
2.0             # mass
-2.0 2.0 1999   # xMin xMax nPoint
1 3             # first and last eigenvalue in output
linear          # interpolation type
6               # nr. of interpolation points and xy declarations
-2.0   0.0
-0.5   0.0
-0.5 -10.0
 0.5 -10.0
 0.5   0.0
 2.0   0.0
```



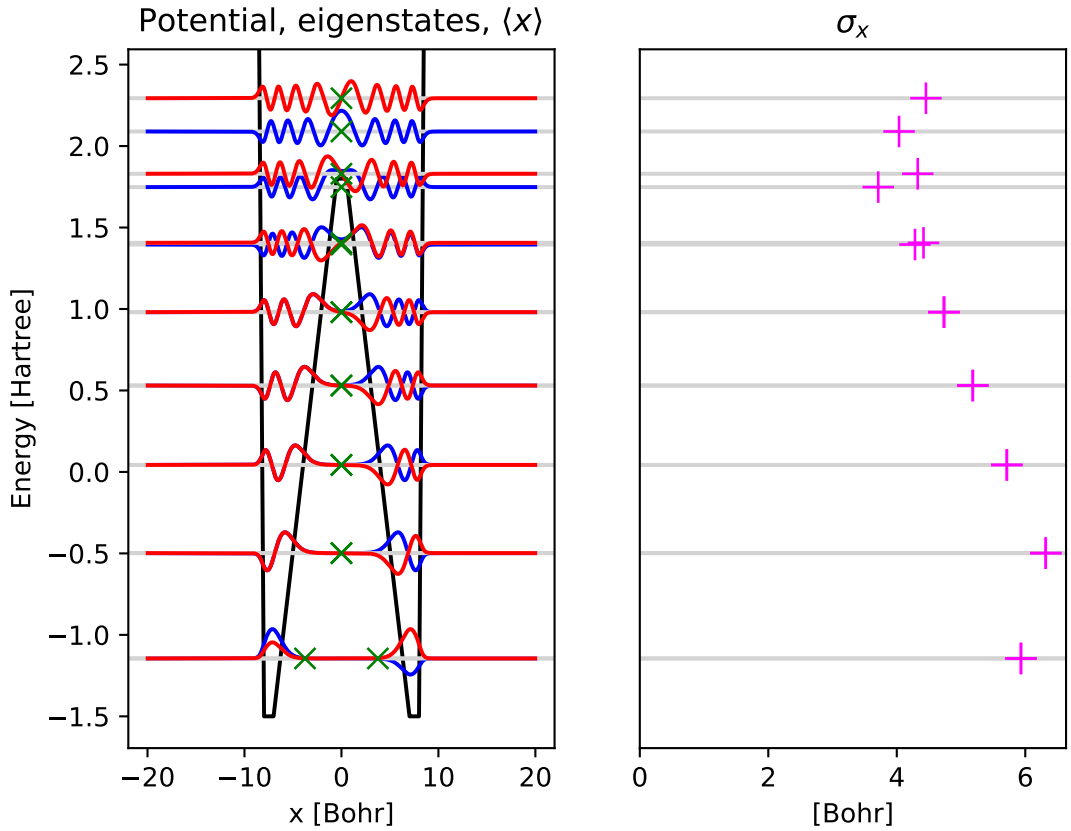
5.3 Harmonischer Oszillator

```
4.0             # Mass
-5.0 5.0 1999   # xMin xMax nPoint
1 5             # first and last eigenvalue to include in the output
polynomial      # interpolation type
3               # # nr. of interpolation points and xy declarations
-1.0  0.5
 0.0  0.0
 1.0  0.5
```



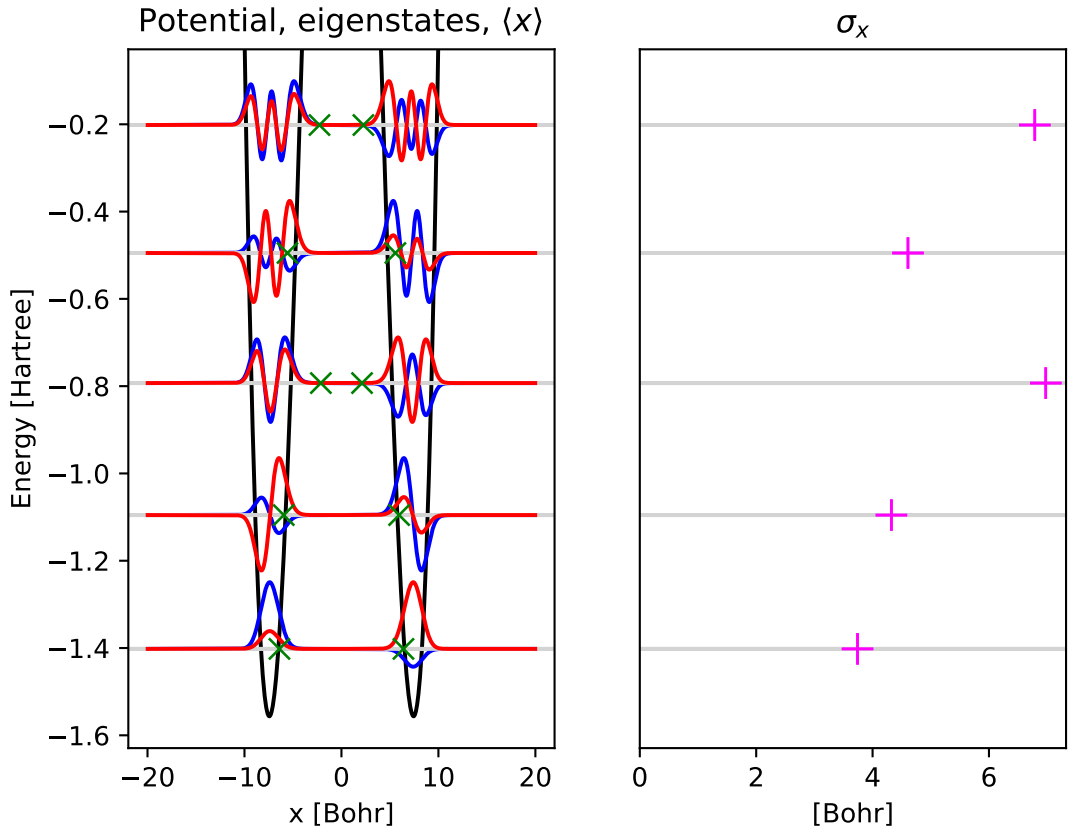
5.4 Doppelter Potentialtopf (linear)

```
2.0                 # mass
-20.0 20.0 1999     # xMin xMax nPoint
1 16                # first and last eigenvalue to print
linear              # interpolation type
8                   # nr. of interpolation points and xy declarations
-20.0  100.0
 -8.0   -1.5
 -7.0   -1.5
 -0.5    1.8
  0.5    1.8
  7.0   -1.5
  8.0   -1.5
 20.0  100.0
```



5.5 Doppelter Potentialtopf (natürlicher kubischer Spline)

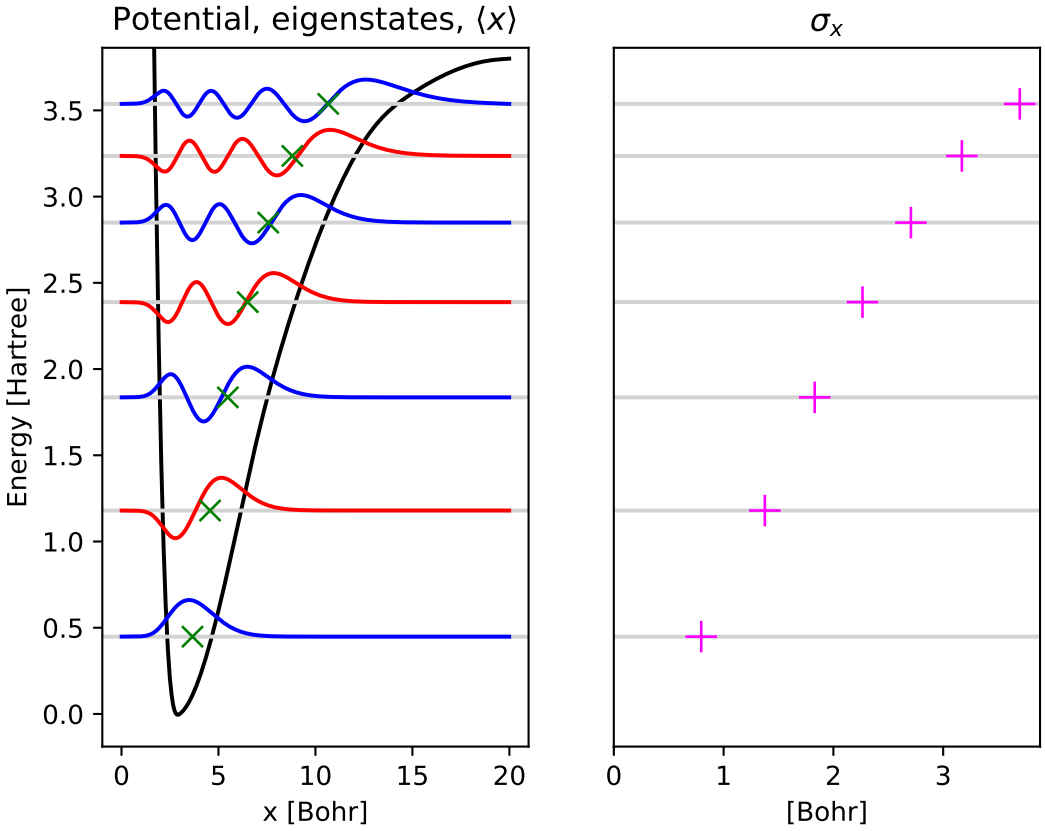
```
4.0                 # mass
-20.0 20.0 1999     # xMin xMax nPoint
1 10                # first and last eigenvalue to include in the output
cspline             # interpolation type
5                   # nr. of interpolation points and xy declarations
-20.0  35.0
-10.0   0.0
  0.0   2.0
 10.0   0.0
 20.0  35.0
```



5.6 Asymmetrischer Potentialtopf

```
1.0             # mass
0.0 20.0 1999   # xMin xMax nPoint
1 7             # first and last eigenvalue to include in the output
cspline         # interpolation type
12              # nr. of interpolation points and xy declarations
 0.0  30.0
 1.0  11.8
 2.0   1.7
 3.0   0.0
 5.0   0.6
 7.0   1.6
 9.0   2.4
11.0   3.0
13.0   3.4
15.0   3.6
```

```
19.0    3.79
20.0    3.8
```



6 Mathematische Formulierung

6.1 Die zeitunabhängige Schrödingergleichung

Das Verhalten makroskopischer Teilchen wird in der Quantenmechanik durch eine komplexe Wellenfunktion $\Psi(x)$ beschrieben, die von der Ortskoordinate als Variable abhängt. Das Betragsquadrat dieser Wellenfunktion gibt die Aufenthaltswahrscheinlichkeitsdichte des Teilchens an einem bestimmten Ort an. Im eindimensionalen stationären Fall, wenn also das äußere Potential nicht von der Zeit abhängt, gehorcht die Wellenfunktion der Schrödingergleichung

$$-\frac{1}{2M}\Psi''(x) + V(x)\Psi(x) = E\Psi(x), \tag{1}$$

wobei $\Psi''(x)$ die zweite Ableitung der Wellenfunktion nach dem Ort ist. Die Größen $V(x)$, M und E geben jeweils das Potential bzw. die Masse und die möglichen Energien des Teilchens

an. Da das Betragsquadrat der Wellenfunktion die Aufenthaltswahrscheinlichkeitsdichte ist, muss das Integral der Wellenfunktion auf den gesammten Raum eins ergeben, d.h.

$$\int |\Psi(x)|^2 \, dx = 1. \tag{2}$$

Die Funktion $\Psi(x)$ sollte stetig und stetig differenzierbar sein.

Da es die Gleichungen wesentlich vereinfacht, werden diese in dieser Beschreibung in atomaren Einheiten aufgeschrieben. Bei diesem Einheitssystem wird die Masse in Elektronenmasse (m_e), die Ladung in Einheitsladung (e), die Wirkung in planckscher Konstante durch 2π ($\hbar$) und die Dielektrizität in $1/(4\pi\epsilon_0)$ angegeben, wobei ϵ_0 die Dielektritizitätskonstante ist. Es gilt also die Relation

$$m_\mathrm{e} = e = \hbar = 1. \tag{3}$$

Die Größen in diesem Einheitsystem werden als dimensionslos betrachtet. Die Einheit der Energie ist Hartree, das 27,211385 eV entspricht. Die Einheit der Länge ist Bohr (bohrscher Radius), das $0{,}529177 \times 10^{-10}$ m entspricht.

6.2 Formulierung als Eigenwertproblem einer Matrix

Ein möglicher Weg zur Lösung der eindimensionalen zeitunabhängigen Schrödingergleichung (1) ist die Diskretisierung dieser Differentialgleichung. Die Wellenfunktion $\Psi(x)$ und das Potential $V(x)$ wird auf einem äquidistanten Punktgitter dargestellt:

$$\Psi_i = \Psi(x_i) \qquad \text{und} \qquad V_i = V(x_i) \qquad i = 1, ..., N, \tag{4}$$

wobei x_i die x-Koordinate der Gitterpunkte angibt, und N die Anzahl der Punkte ist. Die Ableitung des Potentials wird mit der Methode der finiten Differenzen gebildet:

$$\Psi_i' = \Psi'(x_i) \approx \frac{\Psi_{i+1} - \Psi_{i-1}}{2\Delta}, \tag{5}$$

$$\Psi_i'' = \Psi''(x_i) \approx \frac{\Psi_{i+1} - 2\Psi_i + \Psi_{i-1}}{\Delta^2}, \tag{6}$$

wobei Δ den Abstand der Gitterpunkte angibt.

Mit Hilfe dieser Diskretisierung lässt sich die Schrödingergleichung folgendermaßen umformen:

$$-\frac{1}{2M}\Psi_i'' + V_i\Psi_i = E\Psi_i \tag{7}$$

$$-\frac{1}{2M}\left(\frac{\Psi_{i+1} - 2\Psi_i + \Psi_{i-1}}{\Delta^2}\right) + V_i\Psi_i = E\Psi_i \tag{8}$$

$$-\frac{1}{2M\Delta^2}\Psi_{i-1} + \left(\frac{1}{M\Delta^2} + V_i\right)\Psi_i - \frac{1}{2M\Delta^2}\Psi_{i+1} = E\Psi_i. \tag{9}$$

Das Problem der Eigenfunktionsuche eines Operators wird somit in das viel einfachere Problem der Eigenvektorsuche einer Matrix transformiert. Legt man nämlich die Randbedingungen so fest, dass die Wellenfunktion an den Rändern des diskretisierten Raumbereiches verschwindet ($\Psi_0 = \Psi_{N+1} = 0$), so kann man Gleichung (9) für alle Punkte ($i = 1, ..., N$) gleichzeitig in der Matrixform

$$\begin{pmatrix} a+V_1 & -\frac{1}{2}a & 0 & \dots & 0 \\ -\frac{1}{2}a & a+V_2 & -\frac{1}{2}a & \dots & 0 \\ 0 & -\frac{1}{2}a & a+V_3 & \dots & \vdots \\ \vdots & \vdots & & \ddots & -\frac{1}{2}a \\ 0 & 0 & \dots & -\frac{1}{2}a & a+V_N \end{pmatrix} \begin{pmatrix} \Psi_1 \\ \Psi_2 \\ \vdots \\ \Psi_N \end{pmatrix} = E \begin{pmatrix} \Psi_1 \\ \Psi_2 \\ \vdots \\ \Psi_N \end{pmatrix} \tag{10}$$

aufschreiben, wobei zur Abkürzung die Größe $a = \frac{1}{M\Delta^2}$ eingeführt wurde.

Die quadratische Matrix in dieser Eigenwertgleichung hat dank der gewählten Randbedingungen eine tridiagonale Struktur, da sie nur in der Hauptdiagonale und in der ersten oberen und in der ersten unteren Nebendiagonale von Null verschiedene Elemente enthält. Ferner ist diese Matrix auch symmetrisch. Für Eigenwertprobleme dieser Art gibt es sehr effiziente Algorithmen, die in diversen mathematischen Bibliotheken implementiert worden sind. In SciPy kann z.B. die Funktion `scipy.linalg.eigh_tridiagonal()` dafür verwendet werden, die dann die entsprechende Funktion aus der LAPACK-Bibliothek aufruft.

Die gewählten Randbedingungen haben zur Folge, dass das Programm kein freistehendes System modelliert, sondern ein solches, das zwischen zwei unendlich hohen Barrieren eingeschlossen ist. Man muss sich deshalb immer vergewissern, dass die Randbedingungen das modellierte System nicht beeinflussen. Klingen die Wellenfunktionen schon weit von den Rändern von sich ab, stimmen die Ergebnisse mit jenen für ein freistehendes System überein. Wird jedoch das Verschwinden der Wellenfunktionen an den Rändern durch die Randbedingungen erzwungen, werden die Ergebnisse von denen eines freistehenden Systems abweichen.

6.3 Berechnung von abgeleiteten Größen

Nachdem die (in unserem Fall reellen) Eigenfunktionen als Eigenvektoren der Hamiltonmatrix berechnet worden sind, können weitere abgeleitete Größen, wie z.B. die Erwartungswerte der Ortsoperatorpotenzen, berechnet werden. Die dazu nötigen Integrale der Wellenfunktionen können in einfachster Näherung mit einer diskreten Summation approximiert werden. Dabei sollten die beiden Randpunkte (in denen die Wellefunktion definitionsgemäß verschwindet)

nicht berücksichtigt werden. So ergeben sich zum Beispiel folgende Relationen:

$$
\begin{aligned}
||\Psi||^2 &= \int_{x_0}^{x_1} |\Psi(x)|^2 dx \approx \Delta \sum_i |\Psi_i|^2 \\
\langle \hat{x} \rangle &= \langle \Psi | \hat{x} | \Psi \rangle = \int_{x_0}^{x_1} \Psi(x) x \Psi(x) dx \approx \Delta \sum_i \Psi_i x_i \Psi_i \\
\langle \hat{x}^2 \rangle &= \langle \Psi | \hat{x}^2 | \Psi \rangle = \int_{x_0}^{x_1} \Psi(x) x^2 \Psi(x) dx \approx \Delta \sum_i \Psi_i x_i^2 \Psi_i
\end{aligned}
$$

Bitte beachten Sie, dass die Berechnung der entsprechenden Erwartungswerte normierte Wellenfunktionen voraussetzt. Es ist also wichtig, dass die diskretiesierten Wellenfunktionen vorher entsprechend normiert werden. (Die Eigenvektoren des Eigenwertproblems, wie sie aus den LAPACK-Routinen zurückkommen, sind normalerweise anders genormt.)