Computerpraktikum zur Quantenphysik
Sommersemester 2019

Dr. Wolfgang Niedenzu
Prof. Dr. Helmut Ritsch

Aufgabe 1: **Die Programmiersprache Julia**

- Offizielle Dokumentation: `https://docs.julialang.org`
- Introducing Julia: `https://en.wikibooks.org/wiki/Introducing_Julia`
- Installation von Julia am ZID-PC (Windows 10):
  - **Achtung: Aufgrund der Konfiguration der ZID-PCs werden die Programme sowie deren Konfigurationsdateien leider nicht global sondern nur lokal am jeweiligen Rechner gespeichert. Stellen Sie daher bitte sicher, dass Sie fürs Praktikum immer den selben Rechner verwenden.**
  - Von `https://julialang.org/downloads/` die 64-bit Windows-Version von Julia v1.1 herunterladen und installieren.
  - Julia starten und mit `]` den Paketmanager öffnen. Dort
    - `add PyPlot`
    - `add PyCall`
    - `add QuantumOptics`
    - `add DifferentialEquations`
    - `add FFTW`

    eingeben, um die entsprechenden Pakete zu installieren (dauert etwas). Anschließend mit `Backspace` den Paketmanager verlassen und mit
    - `using PyPlot`
    - `using QuantumOptics`
    - `using DifferentialEquations`

    die jeweiligen Pakete kompilieren (bei `using PyPlot` werden noch einige Python-Pakete nachinstalliert).
- Um Animationen speichern zu können, muss noch `ffmpeg` installiert werden. Eine Möglichkeit ist, mit

```
conda config --append channels conda-forge
conda install ffmpeg
```

in der Anaconda-Shell das entsprechende Python-Paket hinzuzufügen.

- Installation des Atom-Editors am ZID-PC (Windows 10):
  - Atom von `https://atom.io/` herunterladen und installieren.
  - Im Atom-Editor »File« → »Settings« → »Install« aufrufen und dort das Paket `uber-juno` installieren (dauert etwas).
  - Falls das Julia-Binary nicht erkannt wird: »Packages« → »Julia« → »Settings« und dort den »Julia Path« ensprechend anpassen (sollte allerdings nicht notwendig sein).
  - Die Julia-Konsole wird mit »Packages« → »Julia« → »Open Console« in Atom eingebunden.
  - Hinweis: Mit »Packages« → »Command Palette« → »Toggle« → `Julia: Standard Layout` kann auf das Standard-Juli-Layout für Atom wieder hergestellt werden.
  - Hinweis: Sollte das dunkle Standarddesign nicht zusagen, so lässt es sich unter »File« → »Settings« → »Themes« ändern.
  - Nützliche Tastenkombination: Ganzes Julia-Skript ausführen: `Strg+Shift+Enter`; markierten Bereich ausführen: `Shift+Enter`.

1. Hilfe: `?befehl`
2. Julia-Shell (REPL, read-eval-print loop) verlassen: `exit()` oder `Strg+D`.
3. Pakete hinzufügen: Der Paketmanager wird mit `]` aufgerufen.

   ```
   add QuantumOptics
   ```

4. Variablen:
   - `a=3`
   - `b=5.3`
   - `c=a+b`
5. Arrays:
   - Veranschaulichen Sie sich den Unterschied zwischen
     - `a=[1,2,3,4,5,6]`
     - `a=[1 2 3 4 5 6]`
     - `a=[1,2,3.0,4,5,6]`
     - `a=[1:1:6;]`
     - `a=[1 2 3.0;4 5 6]`

- Auf Elemente zugreifen: `a[2,3]` oder `getindex(a,2,3)`
- Elemente hinzufügen: `push!(a,7.3)`
- Was passiert, wenn Sie nach der Zuweisung

  ```
  b=a
  ```

  a verändern?
- Typen: `a=[1,2,3]` und `a[3]=2.7` oder `b=[1.0,2.0,3.0]` und `b[3]=2+3im` führen zu einem Fehler.

6. Operatoren und elementweise Operatoren:

   ```
   a=[1.,2,3;7,9,3;5,3,1]
   sin(a)
   sin.(a)
   ```

   Beachten Sie den Unterschied zwischen

   ```
   a=[1 2;3 4]
   b=[5 6;7 8]
   a*b
   a.*b
   ```

7. Schleifen

   ```
   z=ComplexF64[]
   r=Float64[]
   phi=Float64[]
   for k=[1:4;]
       temp=sqrt(k)+im*log(k)
       push!(z,temp)
       push!(r,abs(temp))
       push!(phi,angle(temp))
   end
   ```

   Kurzform der `for`-Schleife:

   ```
   x=[sqrt(k) for k=1:4]
   x=[sqrt(k)*l for k=1:4, l=1:10]
   ```

8. Bedingungen:

   ```
   if x>0
       y=sqrt(x)
   elseif x<-5
       y=cos(x)
   else
       y=0
   end
   ```

9. Funktionen

```
function myfunction(x,y=1.0)
    return x*y
end
```

In-Place:

```
function myfunction2(x,y,z)
    x.+=y*z
end
```

Typen:

```
function myfunction3(x::Array,y,z)
    x.+=y*z
end
```

10. Eigenwerte und Eigenvektoren

```
using LinearAlgebra
A=rand(3,3)+im*rand(3,3)
eigenvalues,eigenvectors=eigen(A)
```

Überprüfen Sie mit `inv(eigenvectors)*A*eigenvectors`, dass die Matrix der Eigenvektoren die Matrix A diagonalisiert.

```
using LinearAlgebra
A=rand(3,3)+im*rand(3,3)
A=A+A'
eigenvalues,eigenvectors=eigen(A)
```

Überprüfen Sie, dass die Matrix der Eigenvektoren unitär ist und dass `eigenvectors'*A*eigenvectors` die Matrix A diagonalisiert.

11. Graphische Ausgabe

```
using PyPlot
pygui(true)
x=[0:.01:2;]*pi
y=sin.(x)
figure(1)
clf()
plot(x/pi,y)
xlabel(L"x/\pi")
ylabel(L"sin(x)")
grid()
```

12. Einfache Animation

```
using PyPlot
using PyCall
using Printf
anim=pyimport("matplotlib.animation")
pygui(true)

fig=figure(1)
clf()
x=[0:.01:2;]*pi
y=sin.(x)
function animate(k)
    k+=1
    clf()
    plot(x,y.*k)
    ylim(-105,105)
    title("k="*@sprintf("%03.2f",k))
    xlabel(L"x")
    ylabel(L"k\sin(x)")
    grid()
end
movie=anim.FuncAnimation(fig,animate,frames=100,repeat=false,interval=20)
```

Die Animation kann dann mit `movie.save("animation.mp4")` gespeichert werden (benötigt `ffmpeg`).

13. Ein Beispiel, in dem viele der oben aufgeführten Konzepte verwendet werden können, ist die numerische Integration der Differentialgleichung

$$\ddot{x} = -\omega^2 x \tag{1}$$

des harmonischen Oszillators und die graphische Ausgabe der Lösung. Der Fokus soll dabei dabei mehr darauf liegen, sich mit der Programmiersprache Julia vertraut zu machen, als einen möglichst eleganten und effizienten Code zu schreiben.

Die Differentialgleichung (1) zweiter Ordnung ist äquivalent zum System

$$\begin{pmatrix} \dot{x} \\ \dot{p} \end{pmatrix} = \begin{pmatrix} 0 & \frac{1}{m} \\ -m\omega^2 & 0 \end{pmatrix} \begin{pmatrix} x \\ p \end{pmatrix}. \tag{2}$$

zweier gekoppelter Differentialgleichungen erster Ordnung. Im Euler-Verfahren für die Differentialgleichung

$$\dot{y} = f(y,t) \tag{3}$$

wird die Lösung $y(t_i)$ durch die Werte y_i an den diskreten Zeitpunkten $t_i = t_0 + i\Delta t$ approximiert, wobei

$$y_{i+1} = y_i + f(y_i, t_i)\Delta t. \tag{4}$$

$\dot{x} = \frac{p}{m}$

$\dot{p} = -m\omega^2 x$

Lösen Sie die Differentialgleichung (2) mit dem Euler-Verfahren sowie mit einem Verfahren höherer Ordnung mit adaptiver Schrittweite und stellen Sie das Ergebnis im Phasenraum graphisch dar. Vergleichen Sie das Ergebnis mit der analytischen Lösung

$$\begin{pmatrix} x(t) \\ p(t) \end{pmatrix} = \exp\left[\begin{pmatrix} 0 & \frac{1}{m} \\ -m\omega^2 & 0 \end{pmatrix} t\right] \begin{pmatrix} x_0 \\ p_0 \end{pmatrix}. \tag{5}$$

Worin unterscheiden sich die Lösungen? Hinweis: Berechnen Sie die Energie des Oszillators.