Computerphysik Programmiertutorial 8b

Prof. Dr. Matteo Rizzi und Dr. Markus Schmitt - Institut für Theoretische Physik, Universität zu Köln

ILIAS: https://www.ilias.uni-koeln.de/ilias/goto_uk_crs_3862489.html

Github: https://github.com/markusschmitt/compphys2021

Inhalt dieses Notebooks: Lösen gewöhnlicher Differentialgleichungen (und mehr zum Plotten)

Lösen gewöhnlicher Differentialgleichungen (und mehr zum Plotten)

In [1]:

```
using PyPlot
using DifferentialEquations
```

Wir werden das `DifferentialEquations` Paket benutzen, das hier dokumentiert ist.

Gewöhnliche Differentialgleichung erster Ordnung in einer Variable

Allgemein kann eine gewöhnliche Differentialgleichung erster Ordnung geschrieben werden als:

$$\frac{du}{dt} = f(u,p,t)$$

Hier steht p für weitere Parameter und t für die Zeit. Das Ziel ist dann für einen gegebenen Anfangswert $u_0 = u(t_0)$ die zeitabhängige Lösung $u(t)$ auf einem Intervall $[t_0, t_1]$ zu finden. Diese Probleme werden im `DifferentialEquations` Paket als `ODEProblem` definiert und anschließend gelöst. Die Schritte dazu sind:

1. Definieren der DGL: Entspricht dem Definieren einer Funktion $f(u,p,t)$.
2. Festlegen der Anfangswerte u_0, eines Zeitintervalls $[t_0, t_1]$, und der Parameter p (falls vorhanden).
3. Definition eines `ODEProblem` s
4. Lösen der Gleichung.

Beispiel: Radioaktiver Zerfall wird beschrieben durch

$$\frac{du}{dt} = -\gamma u$$

wobei u die Konzentration der Atomsorte ist und γ die Zerfallsrate. Wir betrachten als Beispiel ^{14}C mit der Halbwertszeit von etwa 5730 Jahren.

Schritt 1: Definition der DGL

In [2]:

```
f(u,p,t) = -p * u
```

Out[2]:

```
f (generic function with 1 method)
```

Schritt 2: Festlegen der Anfangswerte, eines Zeitintervalls, und der Parameter

In [10]:

```
u0 = 1.0
tspan = (0.0, 3e4)
p = log(2.0) / 5730
```

Out[10]:

```
0.00012096809433855938
```

Schritt 3: Definieren eines ODEProblems

In [11]:

```
odeProblem = ODEProblem(f, u0, tspan, p)
```

Out[11]:

```
ODEProblem with uType Float64 and tType Float64. In-place: false
timespan: (0.0, 30000.0)
u0: 1.0
```

Schritt 4: Lösen des Problems

In [12]:

```
sol = solve(odeProblem)
```

Out[12]:

```
retcode: Success
Interpolation: automatic order switching interpolation
t: 14-element Vector{Float64}:
     0.0
     0.6075089041308458
     6.682597945439303
    67.43348835852387
   674.9423924893695
  2501.718243101237
  5024.581669012864
  7996.4333957855215
 11547.288170769843
 15539.269067293577
 19976.839052400086
 24772.859882716366
 29893.46047670711
 30000.0
u: 14-element Vector{Float64}:
 1.0
 0.9999265135058383
 0.9991919455133165
 0.9918758799126708
 0.9215976905125147
 0.7388738502350308
 0.5445399629683778
 0.3801030645901141
 0.24737452941851704
 0.15262805167514504
 0.08922893752166641
 0.049951381225538286
 0.026886851402542756
 0.026542560162726155
```

In [8]:

```
sol.u
```

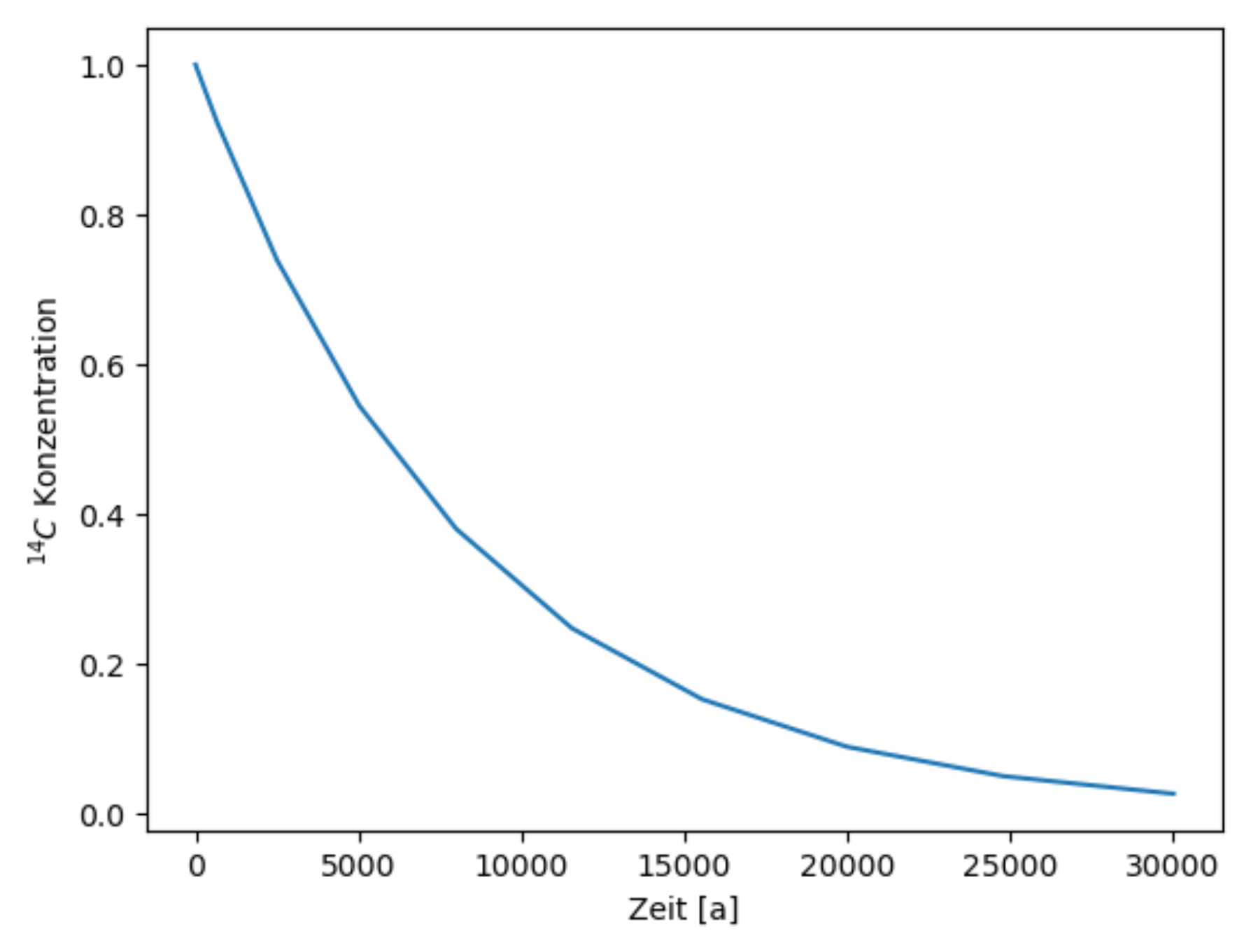
Out[8]:

```
12-element Vector{Float64}:
 1.0
 0.9999265135058383
 0.9991919455133165
 0.9918758799126708
 0.9215976905125147
 0.7388738502350308
 0.5445399629683778
 0.3801030645901141
 0.24737452941851704
 0.15262805167514504
 0.08922893752166641
 0.08897929150683279
```

Plotten des Ergebnisses:

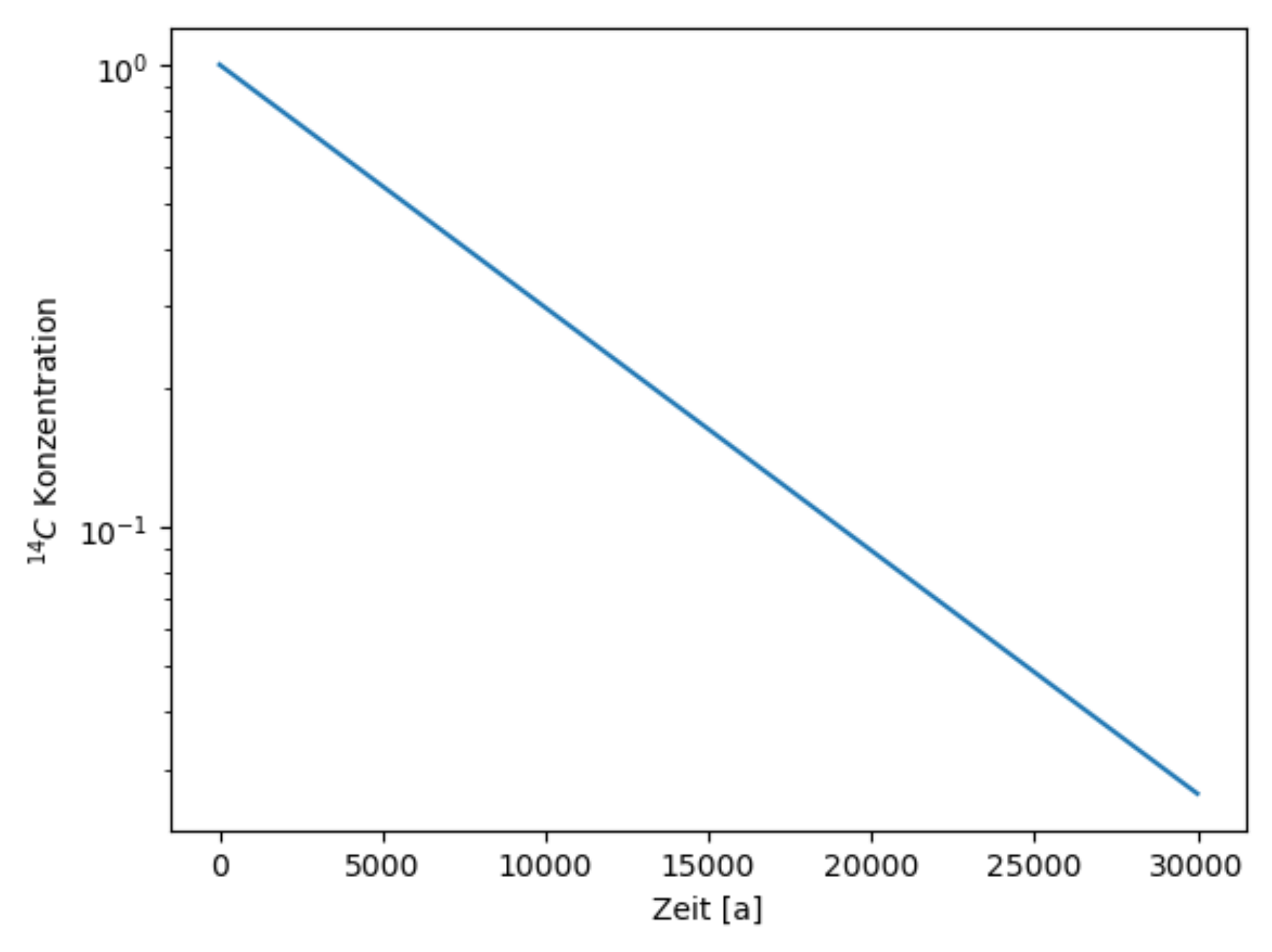
In [13]:

```
plot(sol.t, sol.u)
xlabel("Zeit [a]")
ylabel(L"$^{14}C$ Konzentration");
```



In [14]:

```
semilogy(sol.t, sol.u)
xlabel("Zeit [a]")
ylabel(L"$^{14}C$ Konzentration");
```



Differentialgleichungen höherer Ordnung

Oft interessieren wir uns für DGLs höherer Ordnung, z.b. das Pohl'sche Rad:

$$\ddot{\varphi} + \alpha\dot{\varphi} + \Omega_0\varphi = A\sin(\Omega_E t)$$

Indem wir die Winkelgeschwindigkeit $\omega = \dot{\varphi}$ einführen, können wir diese DGL als ein System von gekoppelten DGLs erster Ordnung schreiben:

$$\begin{aligned}\dot{\varphi} &= \omega\\ \dot{\omega} &= -\alpha\omega - \Omega_0\varphi + A\sin(\Omega_E t)\end{aligned}$$

Problem definieren:

In [15]:

```
function damped_oscillator!(du, u, p, t)
    alpha, Omega0, A, OmegaE = p
    phi, omega = u
    du[1] = omega
    du[2] = - alpha * omega - Omega0 * phi + A * sin(OmegaE*t)
end

u0 = [1,0]
tspan = (0,20pi)
p = [0.2, 1, 0.3, 0.75]

prob = ODEProblem(damped_oscillator!, u0, tspan, p)
```

Out[15]:

```
ODEProblem with uType Vector{Int64} and tType Float64. In-place: true
timespan: (0.0, 62.83185307179586)
u0: 2-element Vector{Int64}:
 1
 0
```

DGL lösen:

In [17]:

```
sol = solve(prob, saveat=0.05)
```

Out[17]:

```
retcode: Success
Interpolation: 1st order linear
t: 1258-element Vector{Float64}:
  0.0
  0.05
  0.1
  0.15000000000000002
  0.2
  0.25
  0.3
  0.35000000000000003
  0.4
  0.45
  0.5
  0.55
  0.6000000000000001
  ⋮
 62.3
 62.35
 62.4
 62.45
 62.5
 62.55
 62.6
 62.65
 62.7
 62.75
 62.8
 62.83185307179586
u: 1258-element Vector{Vector{Float64}}:
 [1.0, 0.0]
 [0.9987590905072455, -0.049449880893219236]
 [0.9950745802552406, -0.09772562792278838]
 [0.9890078811983959, -0.1447200454305796]
 [0.9806256493771663, -0.19033093684337762]
 [0.9699994687651301, -0.23446141370260884]
 [0.9572056409327521, -0.27702005581376343]
 [0.9423250588166678, -0.3179208516518974]
 [0.9254426403551566, -0.3570835582900054]
 [0.9066470174070504, -0.39443395377320495]
 [0.8860305357517336, -0.4299038371187387]
 [0.8636887694967903, -0.4634311647644979]
 [0.8397204569347968, -0.49495983982152036]
 ⋮
 [0.4341151403395139, -0.36161930310880686]
 [0.4157354428974044, -0.37362841814714287]
 [0.3967676130525768, -0.38511074909530385]
 [0.3772380001600272, -0.39604997466813885]
 [0.3571738478229637, -0.4064306858686877]
 [0.336603293892784, -0.41623838598819196]
 [0.315555370469087, -0.42545949060608945]
 [0.2940600038996847, -0.43408132759000917]
 [0.2721480147805753, -0.4420921370957832]
 [0.24985111795596698, -0.449481071567436]
 [0.2272019225182712, -0.45623819573718827]
 [0.21260460560734173, -0.46020926493362746]
```

Plotten mit mehreren Unterplots:

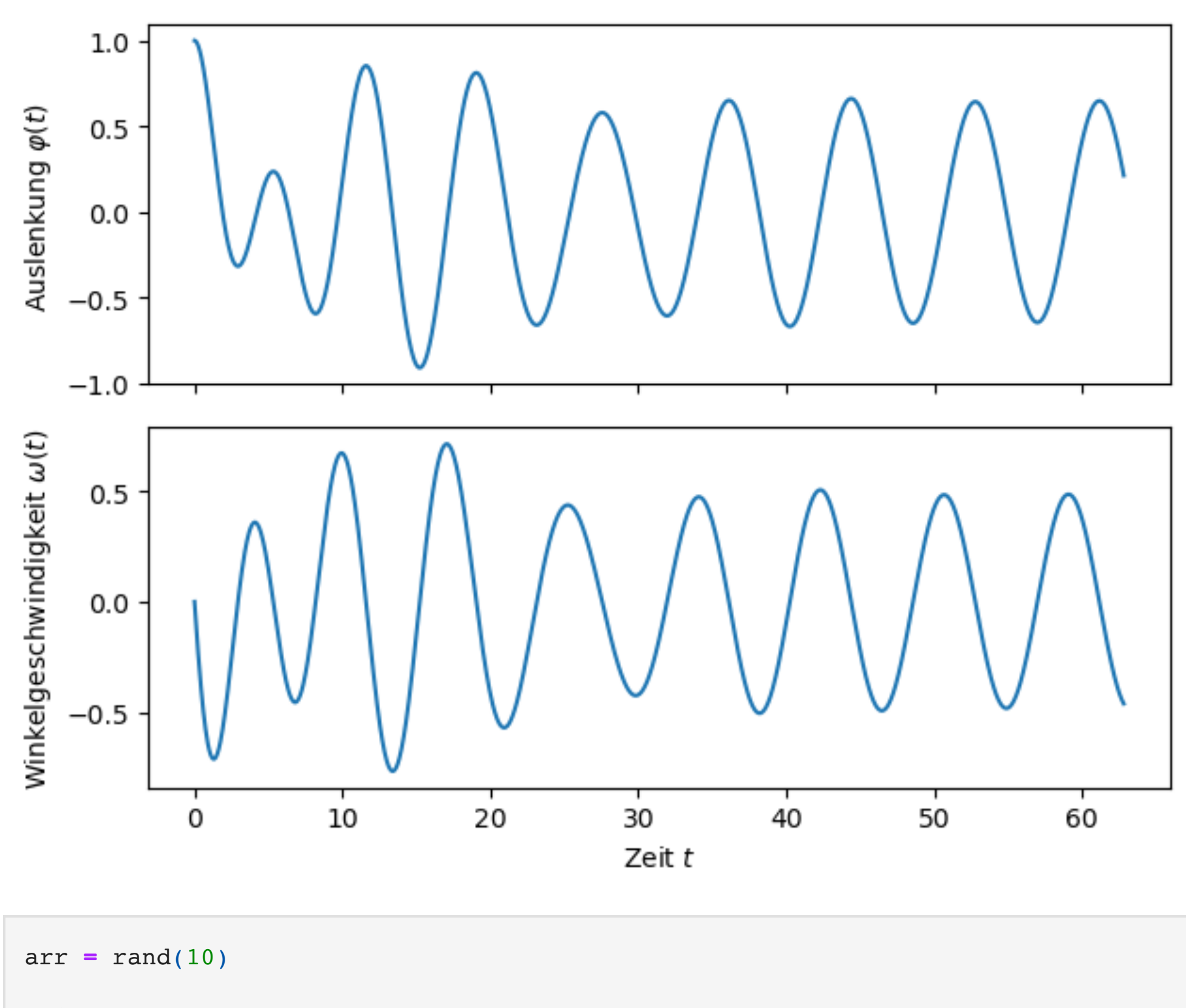
In [19]:

```
fig, ax = subplots(2,1, sharex=true) # Produziere zwei Plots übereinander mit gemeinsamer x-Achse

ax[1].plot(sol.t, getindex.(sol.u,1))
ax[2].plot(sol.t, getindex.(sol.u,2))

# Achsenbeschriftungen
xlabel(L"Zeit $t$")
ax[1].set_ylabel(L"Auslenkung $\varphi(t)$");
ax[2].set_ylabel(L"Winkelgeschwindigkeit $\omega(t)$");

tight_layout()
savefig("oszillator.pdf") # Plot als PDF speichern
```



In [18]:

```
arr = rand(10)

println(arr[3])
println(getindex(arr, 3))
```

```
0.8764265121994377
0.8764265121994377
```

Lorenz-Attraktor

Lorenz-Gleichungen mit Parametern σ, ρ, β:

$$\begin{aligned}\frac{dx}{dt} &= \sigma(y-x)\\ \frac{dy}{dt} &= x(\rho-z)-y\\ \frac{dz}{dt} &= xy-\beta z\end{aligned}$$

Wir definieren ein entsprechendes `ODEProblem` mit festen Parametern $\sigma = 10, \rho = 28, \beta = 8/3$

In [20]:

```
function lorenz!(du,u,p,t)
    du[1] = 10.0*(u[2]-u[1])
    du[2] = u[1]*(28.0-u[3]) - u[2]
    du[3] = u[1]*u[2] - (8/3)*u[3]
end

u0 = [1.0;0.0;0.0]
tspan = (0.0,100.0)
prob = ODEProblem(lorenz!,u0,tspan)

sol = solve(prob, saveat=0.01);
```

Separieren der einzelnen Koordinaten aus dem Lösungsobjekt:

In [21]:

```
ux = getindex.(sol.u,1)
uy = getindex.(sol.u,2)
uz = getindex.(sol.u,3);
```

Plotten in 3D:

In [22]:

```
pygui(true)
plot3D(ux,uy,uz)
```

Out[22]:

```
1-element Vector{PyCall.PyObject}:
 PyObject <mpl_toolkits.mplot3d.art3d.Line3D object at 0x7ff2c0708760>
```