# Computational physics

Jona Ackerschott, Julian Mayr

## Problem set 3

## Problem 1

## Problem 2

To rewrite the schrödinger equation

$$\psi''(z) + \frac{2m}{\hbar^2}(E - V(z))\psi(z) = 0 \tag{1}$$

with the potential $V(z) = mgz$, in a more mathematical manner, the quantities

$$z_0 := \sqrt[3]{\frac{\hbar^2}{2m^2 g}} \qquad E_0 := \sqrt[3]{\frac{mg^2\hbar^2}{2}} \tag{2}$$

with the dimensions length and energy are defined, such that one can substitute $z = z_0\eta$ and gets the following

$$\begin{aligned} & \frac{1}{z_0^2}\psi''(\eta) + \left(\frac{2m}{\hbar^2}E - \frac{2m^2 g}{\hbar^2}z_0\eta\right)\psi(\eta) = 0 \\ \Leftrightarrow \quad & \psi''(\eta) + \left(\frac{2m}{\hbar^2}z_0^2 E - \frac{2m^2 g}{\hbar^2}z_0^3\eta\right)\psi(\eta) = 0 \\ \Leftrightarrow \quad & \psi''(\eta) + (\mathcal{E} - \eta)\,\psi(\eta) = 0 \end{aligned} \tag{3}$$

Where $\frac{2mz_0^2}{\hbar^2} = \frac{1}{E_0}$ was used. This equation is now independent of the units and can be solved numerically with the numerov algorithm. For this one first has to specify the initial conditions. From $V(\eta) = \infty$ for $\eta < 0$ one first can conclude that $\psi(0) = 0$. Second, here $\psi'(0) = 1$ is chosen. In the process the second initial condition is basically arbitrary, because it leads to the same eigenvalues of the energy, and would be set later by the normalization of the wave function. So the specific choice of this value does not matter for this problem, thus it is chosen to be one. Another thing to note here, is that this is probably not the easiest choice for the initial conditions of the numerov algorithm, because one has to first compute the value of $\psi(\Delta t)$, with the step size $\Delta t$ in order to use it (this is done by $\psi(\Delta t) \approx \psi(0) + \psi'(0)\Delta t$), but the derivative at the starting point is the typical choice for an initial condition of an ODE and in this case far more natural and intuitive than to just specify $\psi(\Delta t)$ (It also makes the solution independent of the step size).
Now, with this choices for the initial conditions, one can solve this ODE with the numerov algorithm (where $k(\eta) = \mathcal{E} - \eta$) and the solution will only depend on $\mathcal{E}$. For that, $\Delta t = 0.001$ and $N = 10000$ are chosen.

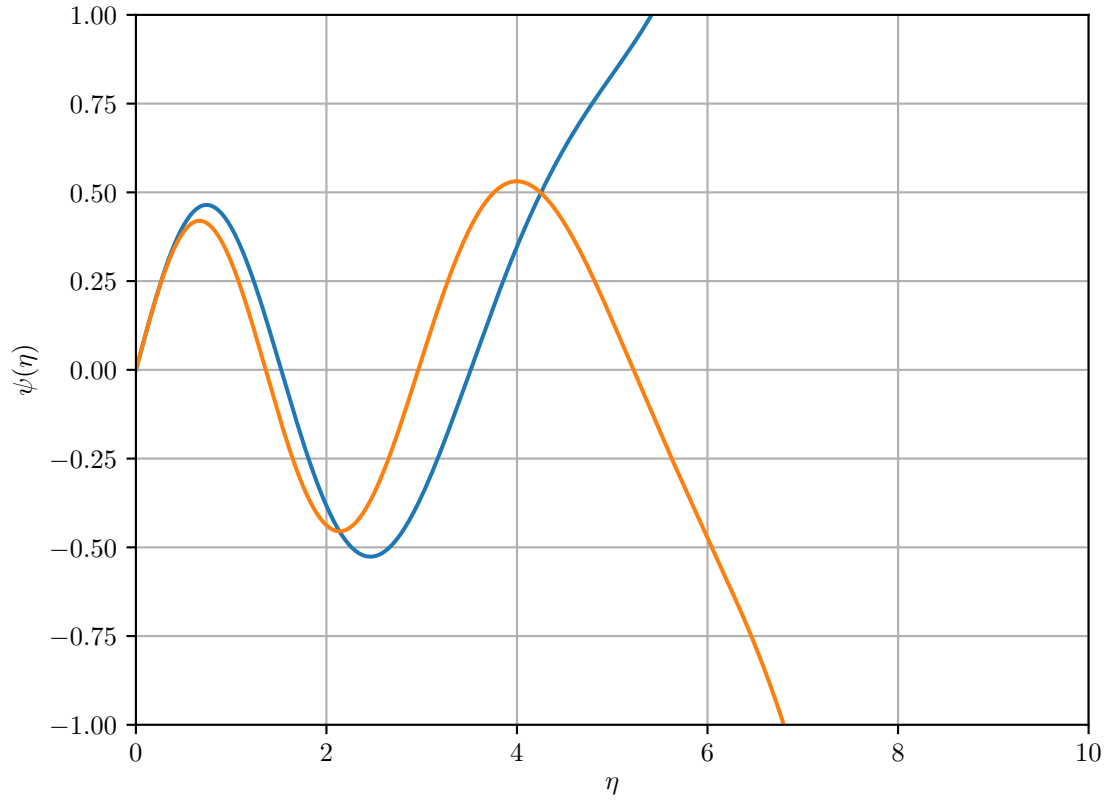


Figure 1: Numerical Solutions of (3) for $\mathcal{E} = 5$ (blue) and $\mathcal{E} = 6$ (orange) with $\psi(0) = 0$ and $\psi'(0) = 1$ as well as $N = 10000$ and $\Delta t = 0.001$

Two typical solutions with different asymptotic behaviour are shown in Figure 1. Generally one gets more „wiggles“ with an increasing value of $\mathcal{E}$, while the solution is constantly changing its asymptotic behaviour with each.
From this one can now determine the energy eigenvalues of this equation, at which the solution is normalizable, meaning it does not diverge at $\eta \to \infty$. One can rephrase this problem by defining a function $\psi_{\text{lim}}(\mathcal{E})$ which returns the value of the solution at $\eta = 10$ (the last $x$-value of the numerical solution, which is not especially large, relatively, but sufficient, because the solution decays exponenially). So to find the eigenvalues of the equation one just has to find the roots of this function. For that purpose the `brentq` method, implemented in `scipy.optimize` is used, which uses brent's method (a combination of the bisection method, the secant method and inverse quadratic interpolation) to determine the roots of a continuous function $f$ in an interval $[a, b]$ which satisfies $f(a) = -f(b)$. For that, the values of $a, b$ for all three cases are determined by trial and error. Such that the solution shows different asymptotic behaviour at $a$ and $b$, and such that the first three eigenvalues (meaning the first three values at which the asymptotic behaviour changes) lay inbetween. This is given by the values $(1, 3), (3, 5)$ and $(5, 6)$. So, with the implemented algorithm one gets the following results for $\mathcal{E}$.

| n | $\mathcal{E}$ |
|---|---|
| 1 | 2.33811 |
| 2 | 4.08795 |
| 3 | 5.52056 |

Table 1: First three eigenvalues of the ODE 3

At last, it is to note, that all the digits of $\mathcal{E}$ in table 1 are accurate (and much more in the output of the code), because with the used algorithm it is no effort to compute the eigenvalues accurate in a moderate amount of time.

### Source Code

```
import numpy as np

def numerov(t0, u0, ud0, k, dt, N):
  """ Numerical numerov algorithm which computes the solution to the ODE u''(t) + k(t)
  t = np.linspace(0, (N - 1) * dt, N)
  u = np.zeros(N)
  u[0] = u0
  u[1] = u0 + ud0 * dt
  for i in range(2, N):
    u[i] = 2 * (1 - 5 * k(t[i - 1]) * dt**2 / 12) * u[i - 1] - (1 + k(t[i - 2]) * dt**2
    u[i] /= (1 + k(t[i]) * dt**2 / 12)
  return t, u
```

Source code 1: numerov.py

```
import matplotlib.pyplot as plt
import numpy as np
import os
from matplotlib.widgets import Slider
from numerov import numerov

# Specific function k used in the schrödinger equation
k = lambda z, eps: eps - z

# The values for eps
eps = np.array([5.0, 6.0])

# Number of steps N and step size dz
N = 10000
dz = 0.001

# Initial valuess
z0 = 0.0
psi0 = 0.0
psid0 = 1.0

# Plot the solutions of the numerov algorithm and save them
plt.axis([0, 10, -1, 1])
for e in eps:
  k_eps = lambda x: k(x, e)
  z, psi = numerov(z0, psi0, psid0, k_eps, dz, N)
  plt.plot(z, psi)
plt.xlabel(r'$\eta$')
plt.ylabel(r'$\psi(\eta)$')
```

```
if not os.path.exists('figures'):
  os.mkdir('figures')
plt.savefig('figures/fig2_1.pgf')
plt.show()
```

Source code 2: problem2a.py

```
import matplotlib.pyplot as plt
import numpy as np
from numerov import numerov
from numpy import tan, pi
from scipy.optimize import brentq

def psi_limit(eps):
  """ Returns the last value of the solution psi """
  k_eps = lambda z: k(z, eps)
  psi = numerov(z0, psi0, psid0, k_eps, dz, N)[1]
  return psi[N - 1]

# Specific function k for the schrödinger equation
k = lambda z, eps: eps - z

# Number of steps N and step size dz
N = 10000
dz = 0.001

# Initial values
z0 = 0.0
psi0 = 0.0
psid0 = 1.0

# Calculate the eigenvalues
eps = np.zeros(3)
bounds = np.array([[1.0, 3.0], [3.0, 5.0], [5.0, 6.0]])
for i in range(len(eps)):
  eps[i] = brentq(psi_limit, *bounds[i])
  print(eps[i])
```

Source code 3: problem2b.py