Exercises IV

Robin Greif (Exercise 3 Francisco Aros), Lia Hankla (Exercise 2 Victor Ksoll)

Due 2018/05/11

1 Exercise 1: Numerov Algorithm for the Schroedinger Equation

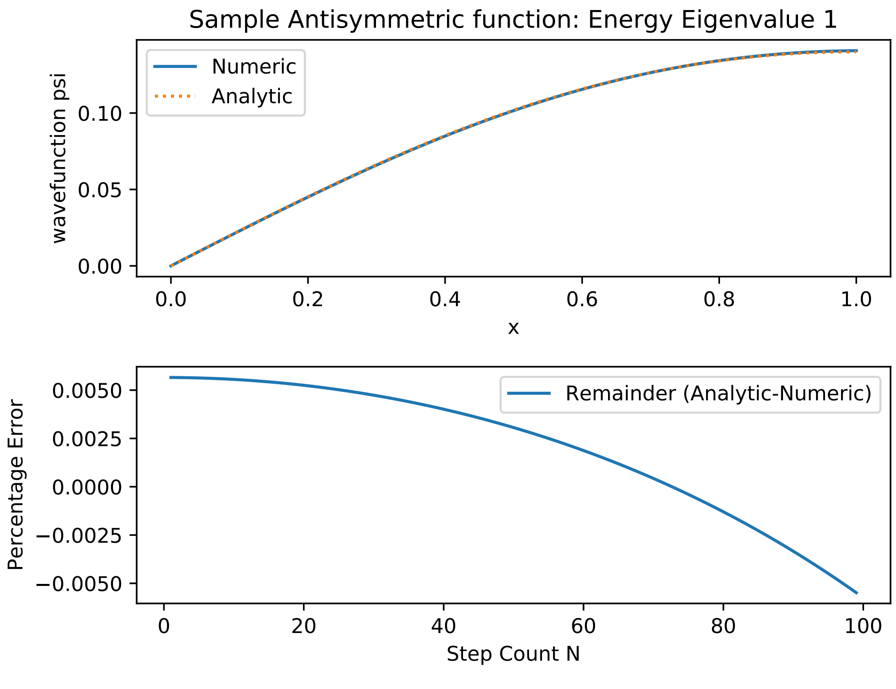

Figure 1: Antisymmetric solutions for harmonic oscillator using the Numerov Algorithm, compared to analytic solution

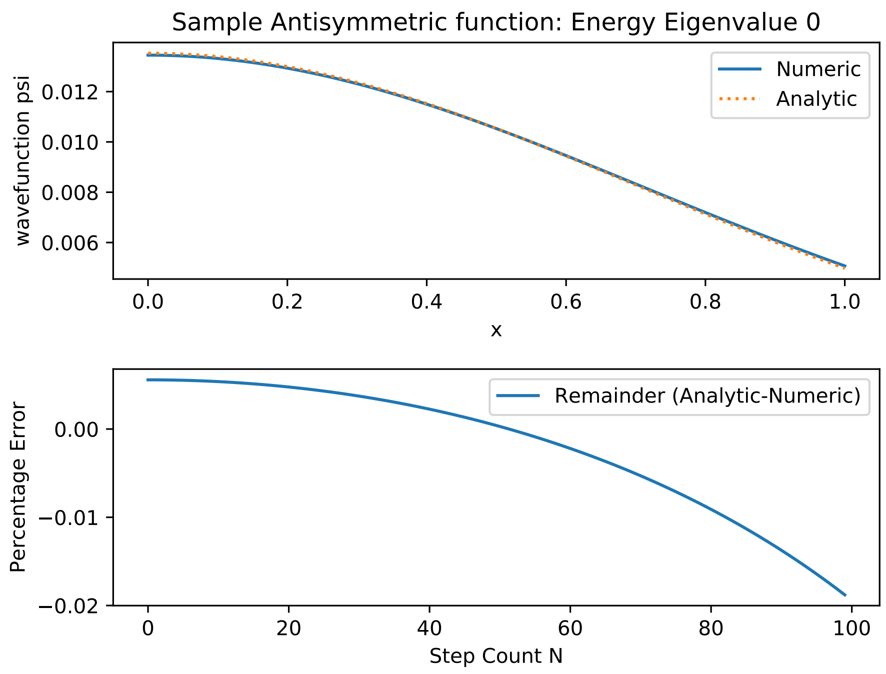

Figure 2: Symmetric solutions for harmonic oscillator using the Numerov Algorithm, compared to analytic solution

Exercise 2: Neutrons in the gravitational field

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

$$0 = \left(\frac{\hbar^2}{2m}\frac{\partial^2}{\partial z^2} - mgz\right)\psi(z) + E\psi(z) \tag{2}$$

$$0 = \left(\frac{\hbar^2}{2m}\frac{1}{R^2}\frac{\partial^2}{\partial z^2} - mgRx\right)\psi(x) + mgR\epsilon\psi(x) \tag{3}$$

$$0 = \psi''(x) + (\epsilon - x)\psi(x) \tag{4}$$

where we used

$$R = \left(\frac{\hbar^2}{2m^2 g}\right)^(1/3) \tag{5}$$

$$x = z/R = V(x) * \left(\frac{1}{mgR}\right) \tag{6}$$

$$\epsilon = \frac{E}{mgR} \tag{7}$$

with units

$$
\begin{aligned}
[R] &= \left(\frac{[Energy \times time]^2}{[mass]^2[length \times time^{-2}]}\right)^{1/3} && (8)\\
&= [Energy^2 \times time^4 \times mass^{-2} \times length^{-1}] && (9)\\
&= [[mass \times length^2 \times time^{-2}]^2 \times time^4 \times mass^{-2} \times length^{-1}] && (10)\\
&= [mass^2 \times length^4 \times time^{-4} \times time^4 \times mass^{-2} \times length^{-1}] && (11)\\
&= [length^3] && (12)\\
[x] &= [length]/[length^3] = [length^{-2}] && (13)\\
[\epsilon] &= \frac{[Energy]}{[mass][m][length \times time^{-2}][length^3]} && (14)\\
&= [Energy \times mass^{-1} \times length^{-1} \times time^2 \times length^{-3}] && (15)\\
&= [mass \times length^2 \times time^{-2} \times mass^{-1} \times length^{-1} \times time^2 \times length^{-3}] && (16)\\
&= [length^{-2}] && (17)
\end{aligned}
$$

Part 1

For large x, these are supposed to approach $\pm\infty$, however, due to some problems in the code, this behavior has not been recovered. Hence, we are unable to plot one of each.

In Fig. 4, various values of ϵ are plotted showing the recovering of sinusoidal behavior without a damping factor, with the frequency increasing as ϵ increases. For $\epsilon < 0$, we recovered an exponential function. This clearly indicates that the potential function applied has a problem, specifically, we have no mirror interactions or gravitational effect visible in the solutions. An explicit integration of the fully specific Equation should recover the true behavior, but has not been feasible to be completed to a reasonable defree within the timeframe, due to missing the forrest for all the trees.

The picture shows clearly a general solution behavior of a second order ordinary differential equation, with varying frequencies of the resulting sinusoidal and varing order of the exponential.

Part 2

Given that, $\psi(x) \to 0$ for $x \to \infty$, and increasing ϵ_n leads to $psi(x)$ changing the sign of its asymptotic behavior, you can determine the first 3 bound states. The first bound states should correspond to the first eigentstates where the wavefunction does not diverge. However, due to the lack of a

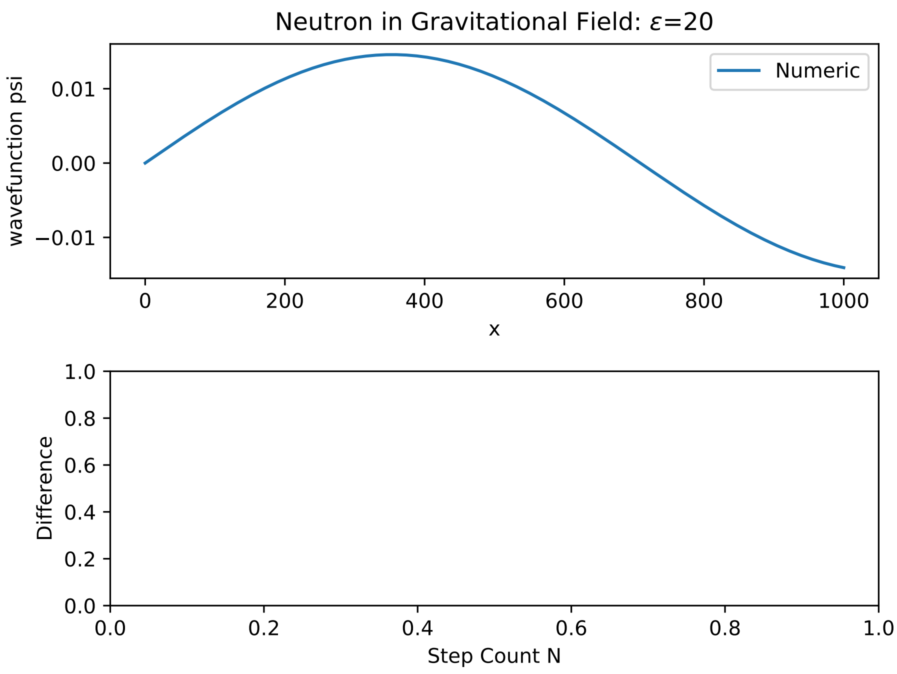

Figure 3: Numerically integrating Eq. 4, plotted well into the classically forbidden range, $x >> \epsilon$

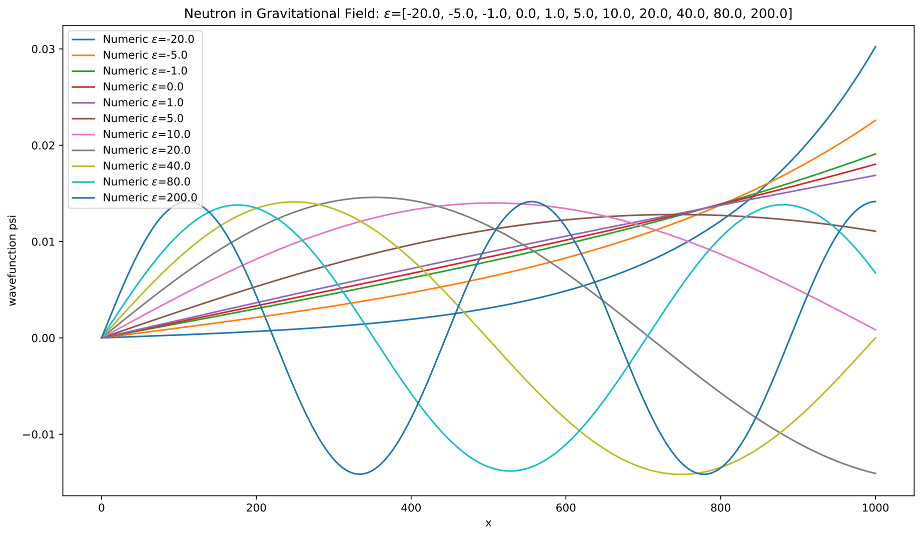

Figure 4: Numerically integrating Eq. 4 with varying ϵ

proper integration we have been unable to show these, and in fact got an continuously spaced solution for general non quantized cases. We must sadly concede defeat at this point. However, for those interested the Diplom of Krantz 2006, has the solutions needed.

Appendix: Code

The code for the exercises is as follows:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import simps


# ------------------------------------------------
#   Exercise 1: Numerov Algorithm for TISE
# ------------------------------------------------

def numerov(psi0, psi1, eps, N, kfun):
    """ Numerov Integration of TISE
    Starting conditions:
      psi0: psi(x=0)
      psi1: psi(x=h)
    eps: normalized energy
    N: number of bins in x direction. Assume start at x=0, go to x = n
    kfun: The function k(x), as in y''(x) + k(x) y(x) = 0
    """

    # The equation has been normalized such that x spans from 0 to 1
    xrange = np.linspace(0, 1, N)

    # Initialize
    psi = np.zeros(N)
    psi[0] = psi0
    psi[1] = psi1

    # Discretize k based on function k(x)
    # Only need to calculate once! Neat.
    kval = kfun(xrange, eps)

    # Step size
    h = 1./N
    # Integrate using the Numerov Algorithm, to O(h^6)
    for i in range(2, N):
        rhs = 2*(1 - (5./12.)*(h**2)*kval[i-1])*psi[i-1] \
              - (1 + (1./12.)*(h**2)*kval[i-2])*psi[i-2]
        lhs = (1 + (1./12.)*(h**2)*kval[i])
        psi[i] = rhs / lhs

    return psi

# kfunc's
def harmonic_oscillator_k(x, eps):
    """" k(x) function as described in Exercise Sheet 4(normalized):
         y''(x) + k(x) y(x) = 0
         x: np.array 1xn
         eps: normalized energy
         This specific function k(x) for harmonic oscillator """
    return 2*eps - x**2

# kfunc's
def eps_mins_x_k(x, eps):
    """" psi''(x) + k(x) psi(x) = 0
         k(x) = (eps-x)
         x: np.array 1xn
         eps: normalized energy """
    #print(np.count_nonzero(x[x<0]), np.max(x), np.min(x))
    ## Never zero...
    x[x<0] = np.inf  # mirror, since x prop to V(z), as V(z)->inf, x->inf
    return eps - x

# analytic tools
def hermite_generator(x, n):
    """ Calculate the nth Hermite polynomial recursively.
        Formula from Exercise Sheet 4. """
    if n == 0:
        return 1
    elif n == 1:
        return 2*x
    else:
        return 2*x*hermite_generator(x, n-1) \
               -2*(n-1)*hermite_generator(x, n-2)
```

```
# analytic tools
def analytic_wavefunction_harmonic(x, n):
    """ Calculate the analytic solution to the TISE
        given that the energy is the nth eigenvalue
        eps = n+1/2 where n is given as an argument.
        x is just the range over which to calculate.
        Formula from Exercise Sheet 4 """
    hermiten = hermite_generator(x, n)
    return hermiten/(2**n * np.math.factorial(n) * np.pi**0.5) * np.exp(-x**2/2.)

def analytic_wavefunction_neutron(x, n):
    """  """
    return


# integral normalization
def normalized_function(psi):
    """ Normalize a function such that the integral of it squared is one.
        Returns the normalized function. """
    psi2 = psi**2
    p2norm = simps(psi2)  # integral over the x direction
    return psi/np.sqrt(p2norm)



# Test odd solution (n = 1, 3, ...)
N = 100  # Steps
xr = np.linspace(0, 1, N)  # Normalize range
n = 1  # degree of Hermite
eps = 1.5
# First values
psi0 = 0
psi1 = 1  # a = 1
# Numeric Solution (normalized)
psi = numerov(psi0, psi1, eps, N, harmonic_oscillator_k)
# Analytic Solution (normalized)
analytic_soln = analytic_wavefunction_harmonic(xr, n)
# Normalize: x, numeric, analytic
normed_psi = normalized_function(psi)
analytic_soln = normalized_function(analytic_soln)
```

```
# Plot 1
fig, axarr = plt.subplots(2,1)
# Solutions
axarr[0].plot(xr, normed_psi, label="Numeric")
axarr[0].plot(xr, analytic_soln,
         linestyle=":", label="Analytic")
axarr[0].legend()
axarr[0].set_xlabel("x")
axarr[0].set_ylabel("wavefunction psi")
axarr[0].set_title("Sample Antisymmetric function: Energy Eigenvalue {}".form
# Remainder
axarr[1].plot((analytic_soln-normed_psi)/analytic_soln, label="Percentage Err
axarr[1].legend()
axarr[1].set_xlabel("Step Count N")
axarr[1].set_ylabel("Percentage Error")
plt.tight_layout()
fig.savefig("exercise4_problem1_antisymEx.pdf")
# plt.show()

# Test even solution (n = 0, 2, ...)
# Note the starting conditions and energy are different
N = 100
xr = np.linspace(0, 1, N)  # Normalize range
n = 0  # degree of hermite
eps = 0.5
# First Values
psi0 = 1
psi1 = psi0 - (1./N)**2*psi0/2*harmonic_oscillator_k(0, eps)
# Numeric Solution
psi = numerov(psi0, psi1, eps, N, harmonic_oscillator_k)
# Analytic Solution
analytic_soln = analytic_wavefunction_harmonic(xr, n)
# Normalize: x, numeric, analytic
normed_psi = normalized_function(psi)**2  #
analytic_soln = normalized_function(analytic_soln)**2
# Plot 2
fig, axarr = plt.subplots(2,1)
# Plot 2: Solutions
axarr[0].plot(xr, normed_psi, label="Numeric")
axarr[0].plot(xr, analytic_soln,
         linestyle=":", label="Analytic")
```

```
axarr[0].legend()
axarr[0].set_xlabel("x")
axarr[0].set_ylabel("wavefunction psi")
axarr[0].set_title("Sample Antisymmetric function: Energy Eigenvalue {}".format(n))
# Plot 2: Remainder
axarr[1].plot((analytic_soln-normed_psi)/analytic_soln, label="Percentage Error (Ana
axarr[1].legend()
axarr[1].set_xlabel("Step Count N")
axarr[1].set_ylabel("Percentage Error")
plt.tight_layout()
fig.savefig("exercise4_problem1_symEx.pdf")





# ----------------------------------------------
#   Exercise 2: Neutrons in the gravitational field
# ----------------------------------------------
# Finding stationary states in the gravitational field of Earth
# https://www.physi.uni-heidelberg.de/Publications/dipl_krantz.pdf


## Part 1:

# a) Solve using Numerov
# Specify length & energy units:
#   eps = E*2m/hbar
#   x = (2m^2 g/hbar)*z = (2m/hbar) V(x)
# Or:
#   R = (hbar^2/(sm^2 g)^1/3
#   x = z/R
#   eps = E/(mgR)
# General
N = 10000
## Numeric Solutions
eps = 20
x_last = 1000  # x_last >> eps
psi0 = 0  # trivial first solution
psi1 = 1  #
```

```
psi = numerov(psi0, psi1, eps, N, eps_mins_x_k)
## Analytic
n = 2
xr = np.linspace(0, x_last, N)
#analytic_soln = analytic_wavefunction_neutron(xr, n)
# Normalize
normed_psi = normalized_function(psi)
analytic_soln = normalized_function(analytic_soln)
# b) Plot solution well into classically forbidden zone
#    that is:  from x=0 to x>>eps  for some values for eps
# Plot 3
fig, axarr = plt.subplots(2,1)
# Plot 3: Solutions
axarr[0].plot(xr, normed_psi, label="Numeric")
#axarr[0].plot(xr, analytic_soln,
#         linestyle=":", label="Analytic")
axarr[0].legend()
axarr[0].set_xlabel("x")
axarr[0].set_ylabel("wavefunction psi")
axarr[0].set_title("Neutron in Gravitational Field: $\epsilon$={}".format(eps
# Plot 3: Remainder
#axarr[1].plot(analytic_soln-normed_psi, label="Remainder (Analytic-Numeric)"
#axarr[1].legend()
axarr[1].set_xlabel("Step Count N")
axarr[1].set_ylabel("Difference")
plt.tight_layout()
fig.savefig("exercise4_problem2_numIntegration.pdf")


# c) for large x: does it approach +/- inf?


# d) Plot two solutions (for two values of eps), one of each

# plotting a bunch, all with the same shape
eps_list = [-20.0, -5.0, -1.0, 0.0, 1.0, 5.0, 10.0, 20.0, 40.0, 80.0, 200.0]
fig, axarr = plt.subplots(1,1, figsize=(12,7))
for eps in eps_list:
    normed_psi = normalized_function(numerov(psi0, psi1, eps, N, eps_mins_x_k
    axarr.plot(xr, normed_psi, label="Numeric $\epsilon$={}".format(eps))
axarr.legend()
axarr.set_xlabel("x")
axarr.set_ylabel("wavefunction psi")
axarr.set_title("Neutron in Gravitational Field: $\epsilon$={}".format(eps_list))
plt.tight_layout()
fig.savefig("exercise4_problem2_varyEps.pdf")

## Part 2:
# eigenvalues, eps_n, belong to normalizable eigenfunctions
# with psi(x)->0 for x->inf
# therefore, increasing eps_n => psi(x) changes sign for x->inf
# a) use this property and eps_n of the first 3 bound states to 2 after comma decima


plt.show()
```