Project 2 in FMNN10

David Engström & Nils Thorin

November 2022

1 2pBVP solvers

Theory

In this project we studied methods of numerically solving 2pBVP, i.e., problems on the form

$$y'' = f(x, y),$$
$$y(0) = \alpha,\ y(L) = \beta.$$

To do this we first discretized the problem by instead of considering y as a continuous function using the $N + 2$ values, $y_i = y(\frac{i \cdot L}{N+1})$, $0 \leq i \leq N + 1$, as an approximation for y. We then used the discrete approximation of the second derivative, $\frac{y_{i+1} - 2y_i + y_{i-1}}{(\Delta x)^2}$ where $\Delta x = \frac{1}{N+1}$ is the distance between adjacent points. Using this approximation our original problem became

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{(\Delta x)^2} = f(x_i, y_i) \qquad i \in [1, N] \tag{1}$$

$$y_0 = \alpha, \qquad y_{N+1} = \beta \tag{2}$$

where $x_i = \frac{i \cdot L}{N+1}$ is the value of x corresponding to each y_i.

To simplify the problem we then substituted $y_0 = \alpha$ and $y_{N+1} = \beta$ into (1) which left us with N equations containing N unknowns (y_i). We also simplified the equations by limiting ourself to linear systems, i.e., systems were f is independent of y. That way the right hand side could be written as $f(x_i)$ instead of $f(x_i, y_i)$. Using this, and rewriting our equations as in the instructions we got the matrix equation

$$\frac{1}{\Delta x^2} \begin{pmatrix} -2 & 1 & 0 & \cdots & \\ 1 & -2 & 1 & & \\ & 1 & -2 & 1 & \\ & & & \ddots & \\ & \cdots & 0 & 1 & -2 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ \\ y_N \end{pmatrix} = \begin{pmatrix} -\alpha/\Delta x^2 + f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{N-1}) \\ -\beta/\Delta x^2 + f(x_N) \end{pmatrix}.$$

For a given function f (or just the values of f evaluated at the different x_i) this is a simple system of linear equations. As long as a sparse representation of the matrix is used this can be solved by either scipy or matlab with a time complexity of $O(N)$.

Task 1.1

We started task 1.1 by implementing a 2pBVP solver as described above in python. We then tested it by using it to solve the problem

$$y'' = 9e^{3t}$$
$$y(0) = 1, \quad y(1) = e^3.$$

This problem is easily solved analytically, yielding $y = e^{3t}$, but is not simple enough to be solved perfectly numerically (like a linear function would). We could thus compare the exact solution with the numerical solution to see if they matched. As can be seen in figures 1 and 2 the numerical solution approximates the exact solution well when using 2^{14} interior points. This led us to believe that our solution was correct. We then measured how the approximation improved when increasing the number of points. To to this we defined the error as the RMS norm of the difference between the estimated y seen as a vector of y_i:s and the actual solution at the same points. We then calculated the error using different numbers of steps and plotted the error against the number of steps on a log scale. We'd expect the error to be proportional to $(\Delta x)^2$, which means it would be proportional to N^{-2}. To confirm this we plotted both our measurements and the line N^{-2} which resulted in 3. The two two curves seems to be parallel, differing only with a constant, which in a log scale means a different factor, thus our convergence is correct.

The Beam Solver

Task 1.2

In Task 1.2 we were tasked with modelling a beam under load according to the system of equations

$$M'' = q(x) \tag{3}$$
$$u'' = M(x)/(EI) \tag{4}$$

with $q(x) = -50\,\mathrm{kN/m}$, $E = 1.9 \times 10^{11}\,\mathrm{N/m^2}$ and $I(x) = 10^{-3} \cdot (3 - 2\cos^{12}(\frac{\pi x}{L}))$. Since u'' is dependant on M we couldn't treat it as one system. Instead we first used our solver to solve (3) and then used our approximation of M in (4) to get our numerical solution for of u. Doing this with $N = 999$ and plotting u against x yielded 4. As expected this graph shows that the beam shows a very slight deflection downwards with a larger deflection closer to the middle. With $N = 999$ the deflection at the midpoint was $u(5) = -0.011\,741\,059\,085\,879\,973\,\mathrm{m}$.

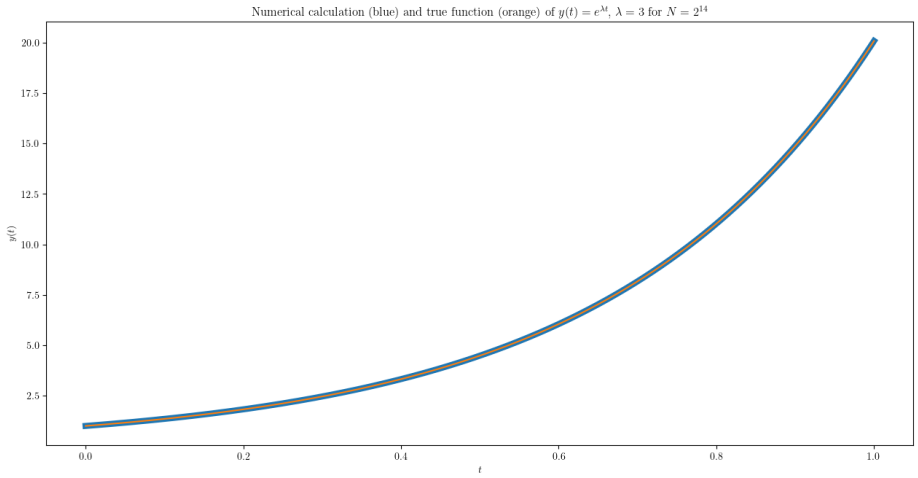

Figure 1: Plot of the function $y(t) = e^{3t}$, both numerically and exactly calculated.

2 Sturm-Liouville eigenvalue problems

Theory

Task 2.1

In task 2.1 we were tasked with solving the eigenvalue problem $u'' = \lambda u$ on the interval $x \in (0, 1)$ with the boundary conditions $u(0) = 0, u'(1) = 0$. As in part 1 we started by discretizing the problem and approximating the second derivative as $u'' \approx \frac{u_{i+1}-2u_i+u_{i-1}}{(\Delta x)^2}$ to give us the equations

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2} = u_i \;\; 1 \leq i \leq N. \tag{5}$$

Including the Dirichlet boundary condition $u(0) = 0$ was done by substituting $u_0 = 0$ into the above equation. To take care of the Neumann boundry condition $u'(1) = 0$ we had to approximate the derivative of u. We did this using the approximation $u'_i = \frac{u_{i-2}-4u_{i-1}+3u_i}{2\Delta x}$ from the slides "chapter3". Since $u'(1)$ corresponds to u'_{N+1} we set $\frac{u_{N-1}-4u_N+3u_{N+1}}{2\Delta x} = u'_{N+1} = 0$. Solving for u_{N+1} this yields $u_{N+1} = \frac{4u_N-u_{N-1}}{3}$ which can then be substituted into (5) leaving us with N equations containing only the N inner points u_i, $i \in [1, N]$. In matrix form this becomes

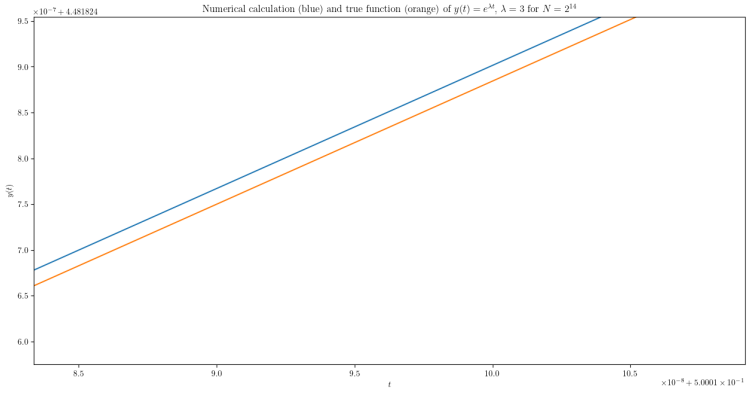

Figure 2: Plot of the function $y(t) = e^{3t}$, both numerically and exactly calculated, zoomed in at around $t = 0.5$.

$$\frac{1}{\Delta x^2}\begin{pmatrix} -2 & 1 & 0 & \cdots & \\ 1 & -2 & 1 & & \\ & 1 & -2 & 1 & \\ & & & \ddots & \\ & & & 1 & -2 & 1 \\ & & \cdots & 0 & 2/3 & -2/3 \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-1} \\ u_N \end{pmatrix} = \lambda \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-1} \\ u_N \end{pmatrix}.$$

(Note that the last row is different because of our substitution).

Our original eigenvalue problem is thus solved by finding eigenvalues and their corresponding eigenvectors of our matrix. To do this we simply used a built in function from scipy. As in task 1.1 we wanted to see how well this solved the problem, and how the solution improved with more points. To do this we compared the three smallest eigenvalues (in magnitude) with the analytical eigenvalues for the problem, found on wikipedia, namely $\lambda_j = -\frac{((2j+1)\pi)^2}{4}$ for $j \in \mathbb{N}$. We calculated these differences for different numbers of points and plotted the error against the number of points used. We also included the line N^{-2} for reference as that is the expected convergence. The result as seen in figure 5 show that our solution approximates the exact solution well, and that our error seems to decrease quadradticaly with an increased amount of points. The eigenvalues we got for $N = 499, \Delta x = 2 \cdot 10^{-3}$ were $\lambda_1 = -2.4673990951820173, \lambda_2 = -22.20644749713279, \lambda_3 = -61.683774387521105$.

We also plotted the first three eigenmodes (corresponding to the three smallest eigenvalues) in figure 6. These seemed to match the theoretical curves, as they were 0 at $x = 0$ and had derivative 0 at $x = 1$.

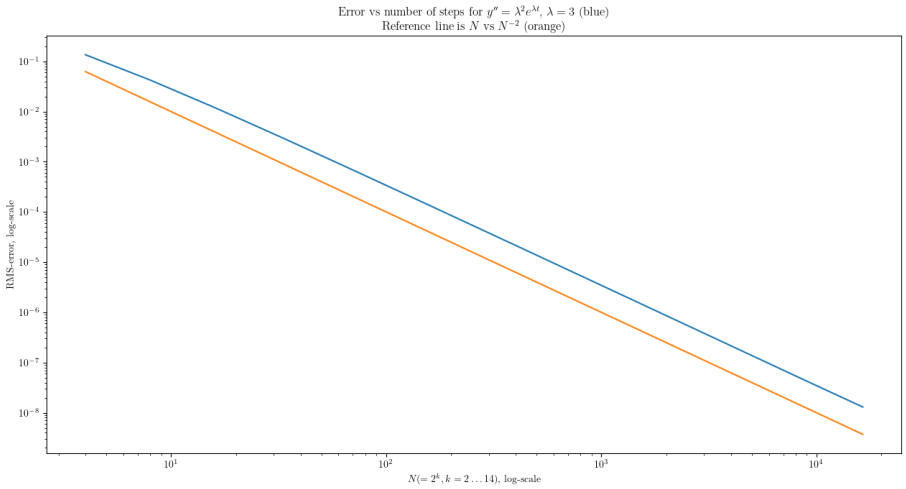

Figure 3: Error against number of steps in a log-log-plot, compared to a line of slope -2.

The Schrödinger equation

Finally, we applied our numerical solver to the Shrödinger equation

$$\psi'' - V(x)\psi = -E\psi \tag{6}$$

where $\psi(x)$ is a wave function, $V(x)$ is a potential function and E is the energy level. To fit this into our solver, we first created a discretization of the second derivative operator as above and subtracted the discretized potential function along the diagonal to create a new operator matrix. We thus got the eigenvalue problem:

$$\frac{1}{\Delta x^2}\begin{pmatrix} 2+V_1 & -1 & 0 & \cdots & & \\ -1 & 2+V_2 & -1 & & & \\ & -1 & 2+V_3 & -1 & & \\ & & & \ddots & & \\ & & -1 & 2+V_{N-1} & -1 & \\ & \cdots & 0 & -1 & 2+V_N & \end{pmatrix}\begin{pmatrix} \psi_1 \\ \psi_2 \\ \psi_3 \\ \vdots \\ \psi_{N-1} \\ \psi_N \end{pmatrix} = E\begin{pmatrix} \psi_1 \\ \psi_2 \\ \psi_3 \\ \vdots \\ \psi_{N-1} \\ \psi_N \end{pmatrix},$$

where $V_i = V(x_i)$.

To test our solver we used it on the problem $\psi'' = -E\psi$. This resulted in the eigenmodes as seen in figure 7. These eigenmodes are standing waves with different frequencies. This is what we would expect which shows that our solver works.

We then tried our solver with the potential function $V(x) = 700(0.5 - |x - 0.5|)$ yielding the eigenmodes shown in figure 8. This potential function introduced a "barrier" in the middle which can be seen in how the lower energy

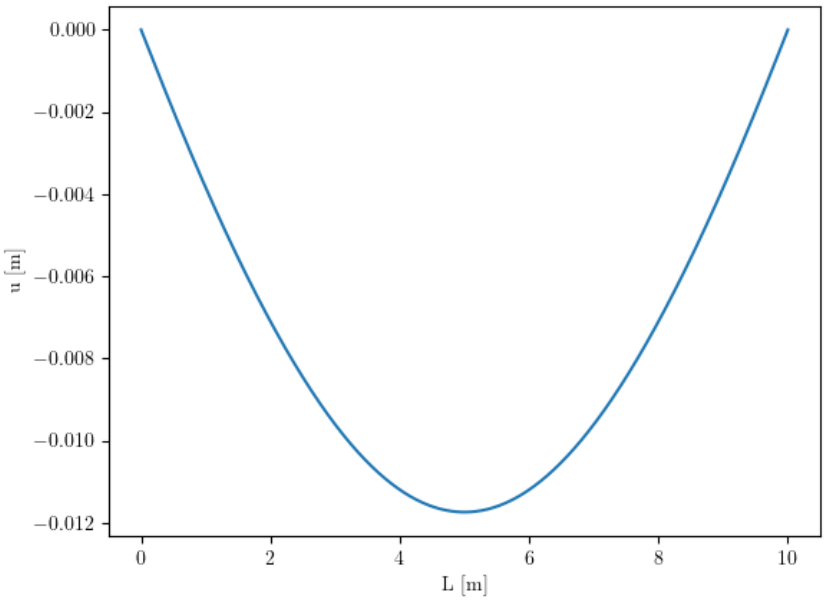

Figure 4: Deflection of a heavy-duty railway flatcar, with $E = 1.9 \times 10^{11}\,\mathrm{N/m^2}$, $I(x) = 10^{-3} \cdot (3 - 2\cos^{12}(\frac{\pi x}{L}))$ and $q = -50\,\mathrm{kN/m}$.

eigenmodes only have peaks to the sides and no peaks in the middle. The first energy level with a significant probability of finding the particle at $x = 0.5$ was a bit over 400, corresponding to the green eigenmode. A double state can also be observed in the left figure where both the black and blue eigenmodes share the energy level of about 200.

We also tried our solver with the potential barrier $V(x) = 800\sin(\pi x)^2$ resulting in a more centralized barrier. The results of this can be seen in figure 9. Many of the effects were similar to with the previous barrier. The lower energy values had peaks only at the sides and double states could be observed for the lowest energy levels. In this case the first energy value with a significant chance of finding the particle at $x = 0.5$ was around 800, corresponding to the green eigenmode. To get three energy wells we used the potential function $V(x) = 800\sin(2\pi x)^2$, as seen in figure 10. Here a triplet state can be observed at the energy level 500.

Finally, for a interesting case, we increased the frequency of the sinus function to 6, meaning we had $V(x) = 800\sin(6\pi x)^2$. This produced many wells, and thereby many wave functions at roughly the same energy levels.

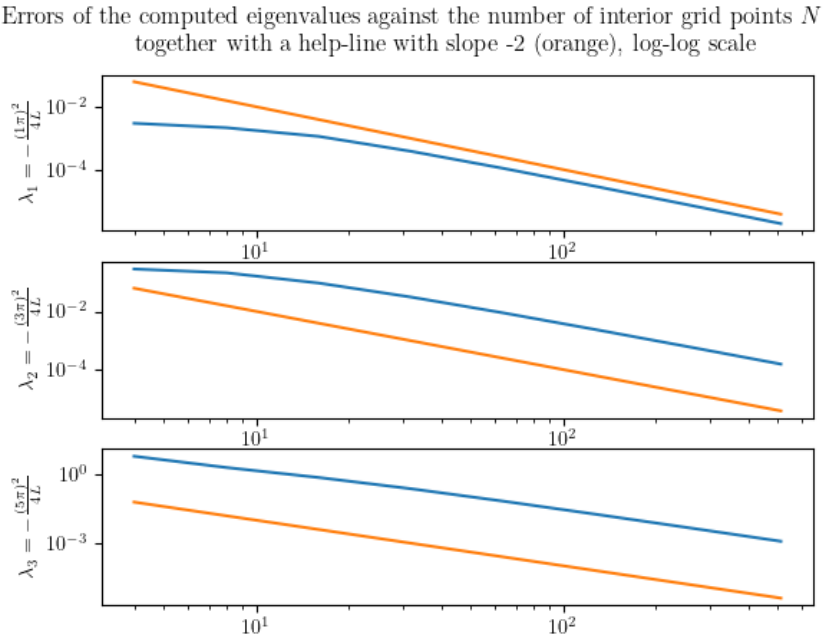

Figure 5: Errors of the first three (smallest in absolute value) eigenvalues against the theoretical eigenvalues $\lambda_i = -\frac{((2i-1)\pi)^2}{4L^2}$.

Conclusion and acknowledgements

To conclude this report we summarize that we have learnt how to discretize the second derivative operator and use it to solve both boundary value problems and Sturm-Liouville problems.

We wish to acknowledge various of our classmates who have helped in comparing the values we found against theirs, as well as those who know more physics than us what the quantum terms meant.

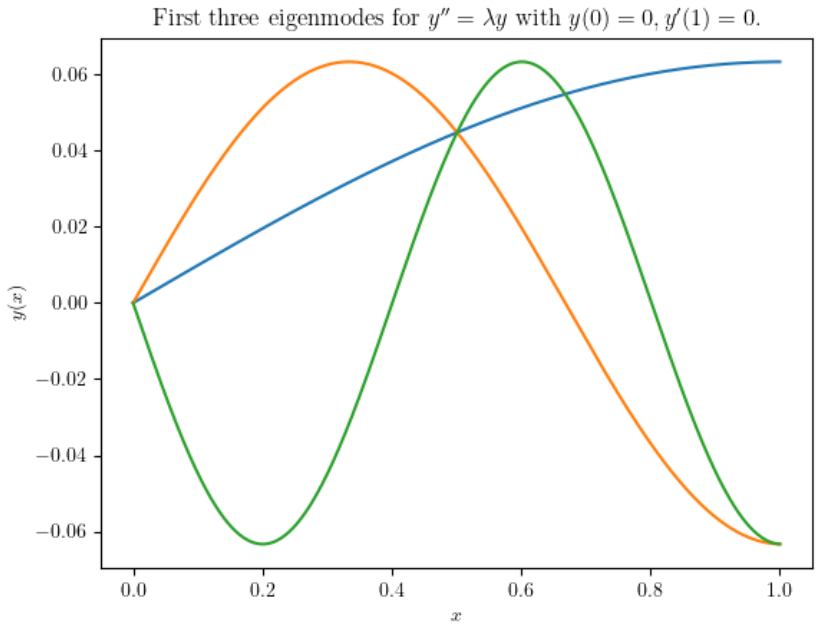

Figure 6: First three eigenmodes for $y'' = \lambda y$ with $y(0) = y'(1) = 0$.

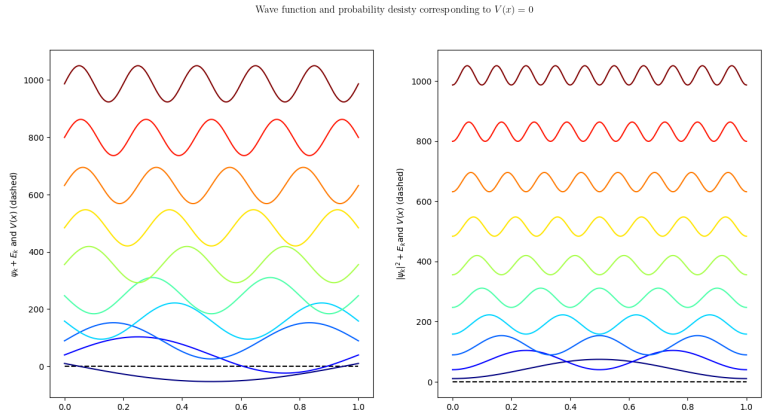

Figure 7: Wave functions and probability densities for $V(x) = 0$.

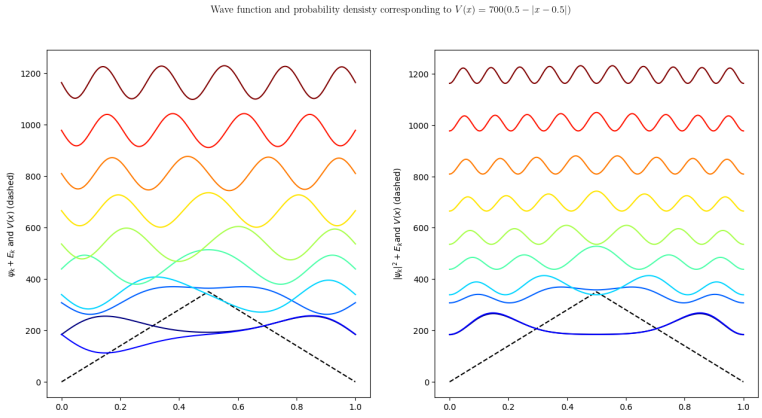

Figure 8: Wave functions and probability densities for $V(x) = 700(0.5-|x-0.5|)$.

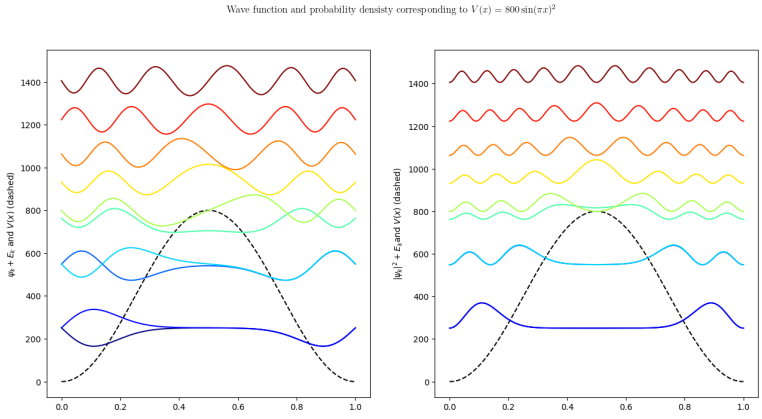

Figure 9: Wave functions and probability densities for $V(x) = 800\sin(\pi x)^2$.

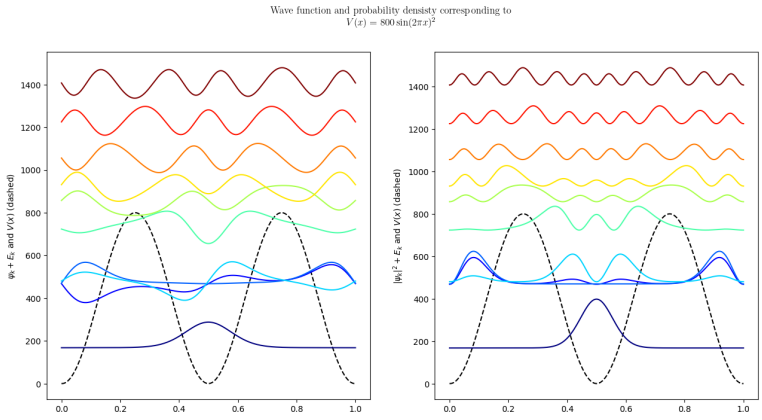

Figure 10: Wave functions and probability densities for $V(x) = 800\sin(2\pi x)^2$.

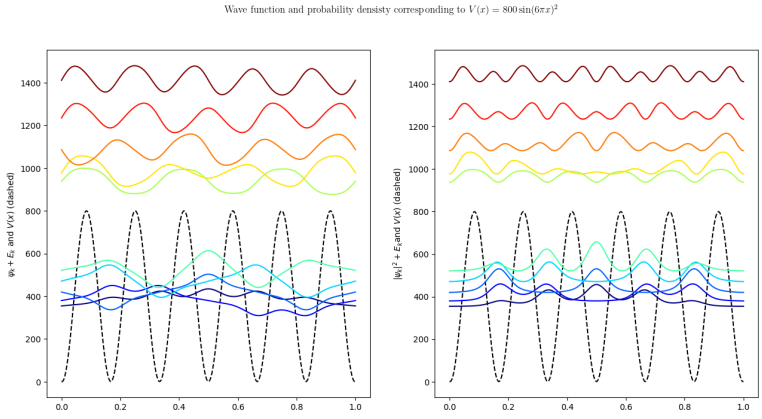

Figure 11: Wave functions and probability densities for $V(x) = 800\sin(6\pi x)^2$.

A Code

Part 1

```
import numpy as np
from scipy.sparse import diags
from scipy.sparse.linalg import spsolve
from scipy.linalg import norm
import math
import matplotlib.pyplot as plt


def twopBVP(fvec, alpha, beta, L, N):
    h = L / (N + 1)
    T = diags([1, -2, 1], [-1, 0, 1], shape=(N, N))
    fvec *= h**2
    fvec[0] += -alpha
    fvec[-1] += -beta
    y = spsolve(T.tocsc(), fvec)
    y = np.insert(y, 0, alpha)
    y = np.append(y, beta)
    return y

def errVSh():
    lambda0 = 3
    f = lambda t: lambda0**2 * np.exp(lambda0 * t)
    y = lambda t: np.exp(lambda0 * t)
    L = 1
    errs = []
    ns = []
    tst = []
    for k in range(2, 15):
        N = 2 ** k
        tgrid = np.linspace(0, L, N + 2)
        fvec = f(tgrid[1:-1])
        ygrid = twopBVP(fvec, y(0), y(1), L, N)
        h = L / (N+1)
        err = math.sqrt(h) * norm(ygrid - y(tgrid)) # using RMS-norm
        errs.append(err)
        ns.append(N)
        tst.append(N ** -2)

    plt.rc('text', usetex=True)
    plt.rc('font', family='serif')
    plt.figure()
    plt.loglog(ns, errs)
```

```
    plt.loglog(ns, tst)
    plt.xlabel("$\displaystyle N (= 2^k, k = 2\ldots 14)$, log-scale")
    plt.ylabel("RMS-error, log-scale")
    plt.title("""Error vs number of steps for $\displaystyle y'' = \lambda^2 e^{\lambda t}$,
                $\lambda = 3$ (blue)\nReference line is $N$ vs $N^{-2}$ (orange)""")

    plt.figure()
    plt.plot(tgrid, ygrid)
    plt.plot(tgrid, y(tgrid))
    plt.xlabel("$t$")
    plt.ylabel("$y(t)$")
    plt.title("""Numerical calculation (blue) and true function (orange) of
                $y(t) = e^{\lambda t}$, $\lambda = 3$ for $N=2^{14}$""")
    plt.show()

def beamSolver():
    N = 999
    L = 10
    E = 1.9 * 10**11
    I = lambda x: 10**-3 * (3 - 2 * np.cos((math.pi * x) / L)**12)
    q = -50 * 10**3 * np.ones(N)
    tgrid = np.linspace(0, L, N + 2)
    M = twopBVP(q, 0, 0, L, N)
    fvec = M[1:-1] / (E * I(tgrid[1:-1]))
    u = twopBVP(fvec, 0, 0, L, N)
    print(u[500]) # -0.011741059085879973
    plt.rc('text', usetex=True)
    plt.rc('font', family='serif')
    plt.figure()
    plt.plot(tgrid, M)
    plt.xlabel("L [m]")
    plt.ylabel("M [Nm]")
    plt.figure()
    plt.plot(tgrid, u)
    plt.xlabel("L [m]")
    plt.ylabel("u [m]")
    plt.show()

if __name__ == '__main__':
    errVSh()
    # beamSolver()
```

Part 2

```
import numpy as np
from numpy.linalg import eig
from scipy.sparse import diags
from scipy.linalg import norm
import math
import matplotlib.pyplot as plt

def SLeig(N):
    L = 1
    h = L / (N + 1)
    T = diags([1, -2, 1], [-1, 0, 1], shape=(N, N))
    T = T.toarray()
    # Set correct values in the double derivate matrix
    # corresponding to homogenous neuman conditions
    # 0 = beta = (y_n-1 - 4y_n + 3y_n+1)/2h
    # ie. y_n+1 = 1/3(4y_n - y_n-1)
    T[-1][-2] = 2/3
    T[-1][-1] = -2/3
    return eig(T/h**2)

def SLplot():
    ns = []
    M = 3
    errs = [[] for _ in range(M)]
    for k in range(2, 10):
        N = 2 ** k
        eigs, modes = SLeig(N)
        tuples = sorted(zip(eigs, modes), key=lambda x: abs(x[0]))
        eigs, modes = [t[0] for t in tuples], [t[1] for t in tuples]
        for j in range(M):
            theoretical = -((2*j + 1) * math.pi)** 2 / 4
            errs[j].append(norm(eigs[j] - theoretical))
        ns.append(N)
    plt.rc('text', usetex=True)
    plt.rc('font', family='serif')
    fig, axs = plt.subplots(3, 1)
    for i in range(M):
        axs[i].loglog(ns, errs[i])
        axs[i].loglog(ns, np.float_power(ns, -2)) # help line at slope -2
        axs[i].set_ylabel("$\lambda_{%d} = -\\frac{(%d \pi)^2}{4L}$" %(i+1, 2*i+1))
    fig.suptitle("""Errors of the computed eigenvalues against the number of
                    interior grid points $N$ (blue)\ntogether with a help-line
                    with slope -2 (orange), log-log scale""")
    plt.show()
```

```
def endpoints(vec):
    vec = np.insert(vec, 0, 0)
    y_final = 1/3 * (4 * vec[-1] - vec[-2])
    return np.append(vec, y_final)

def SL499():
    M = 3
    eigs, modes = SLeig(499)
    tgrid = np.linspace(0, 1, 501)
    sorted_indices = np.argsort(-eigs)[:M]
    n_largest = [(eigs[i], modes[:, i]) for i in sorted_indices]
    plt.rc('text', usetex=True)
    plt.rc('font', family='serif')
    for i, (eig, mode) in enumerate(n_largest):
        print(eig)
        plt.plot(tgrid, endpoints(mode)) # plot with alpha at first index
    plt.xlabel("$x$")
    plt.ylabel("$y(x)$")
    plt.title("First three eigenmodes for $y'' = \lambda y$ with $y(0) = 0, y'(1) = 0$.")
    plt.show()

def pad(vec):
    vec = np.insert(vec, 0, 0)
    return np.append(vec, 0)

def shrodinger(N, vVec):
    h = 1 / (N+1)
    M = 10
    xgrid = np.linspace(0, 1, N + 2)
    T = diags([1, -2, 1], [-1, 0, 1], shape=(N, N))
    V = diags(vVec, 0, shape=(N, N))
    A = (T/h**2-V).toarray()
    eigs, waves = eig(A)
    sorted_indices = np.argsort(-eigs)[:M]
    n_largest = [(eigs[i], pad(waves[:, i])) for i in sorted_indices]
    # Plotting
    fig, axs = plt.subplots(1, 2)
    cm = plt.cm.jet(np.linspace(0, 1, M))
    plt.rc('text', usetex=True)
    plt.rc('font', family='serif')
    for i in range(2):
        axs[i].set_prop_cycle('color', list(cm))
        axs[i].plot(xgrid, pad(vVec), 'k--')
    for (E, wave) in n_largest:
        axs[0].plot(xgrid, 1000 * wave - E)
```

```
        axs[1].plot(xgrid, 16000 * np.power(np.abs(wave), 2) - E)
    fig.suptitle("""Wave function and probability densisty corresponding to
                    $V(x) = 800\sin(2\pi x)^2$""")
    axs[0].set_ylabel("$\psi_k + E_k$ and $V(x)$ (dashed)")
    axs[1].set_ylabel("$|\psi_k|^2 + E_k$ and $V(x)$ (dashed)")
    plt.jet()
    plt.show()

def shrod():
    N = 498
    # V = lambda x: [0 for _ in range(len(x))]
    # V = lambda x: 700 * (0.5 - np.abs(x - 0.5))
    # V = lambda x: 800 * np.power(np.sin(math.pi * x), 2)
    V = lambda x: 800 * np.power(np.sin(2 * math.pi * x), 2)
    xgrid = np.linspace(0, 1, N)
    shrodinger(N, V(xgrid))

if __name__ == "__main__":
    # SLplot()
    # SL499()
    shrod()
```