Project 2

NUMN12 Numerical Methods for Solving Differential Equations

Felicia Segui
Ebba Rickard

November 2019

1 Task 1.1

A two-point Boundary Value Problem solver was created using a finite difference method of the second order. The function was limited to solving only linear BVP depending on x which reduces the method to the linear system in equation 1.[1] The function was created in Matlab and named *twopBVP* (see script in fig 12 in appendix).

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

The function was tested using the simple BVP

$$y'' = 12 \cdot \frac{\beta - \alpha}{L^4} x^2 \qquad y(0) = \alpha, \quad y(L) = \beta$$

which has the analytic solution

$$y = \frac{\beta - \alpha}{L^4} x^4 + \alpha.$$

The solution is of a higher order than two so that the accumulated error can be evaluated properly. [1]

The calculated error is plotted as a function of step size in figure 1 (Sript in figure 13 in appendix). Next to it a function $\propto \Delta x^2$ is graphed for comparison, as the error is expected to have have a square dependency of the step size. The two lines in figure 1 have very similar slopes and the function created is therefore concluded to implement the Finite Different Method properly. This result was also supported by plotting the approximated function values in the same graph as the analytical function values, since the approximation converges to the real solution when the step size grows smaller and approaches zero. This is presented in figure 2a and 2b.

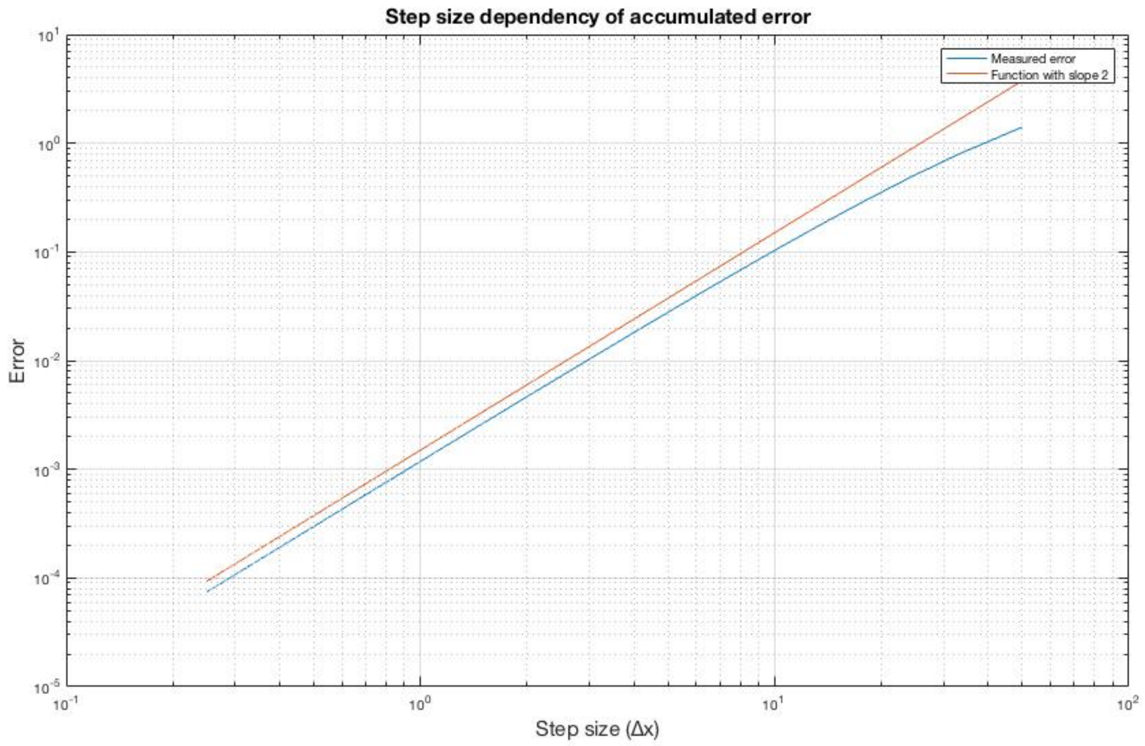
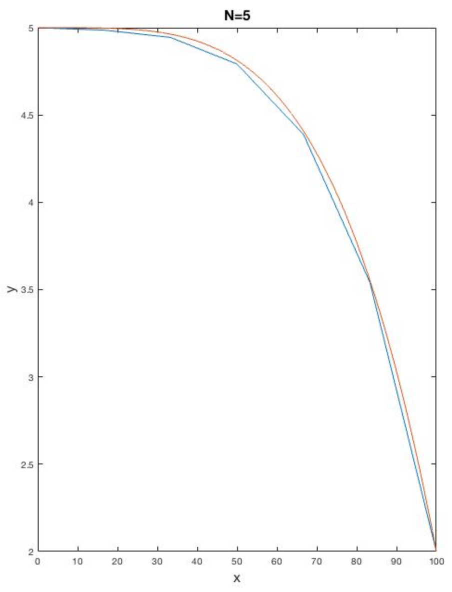
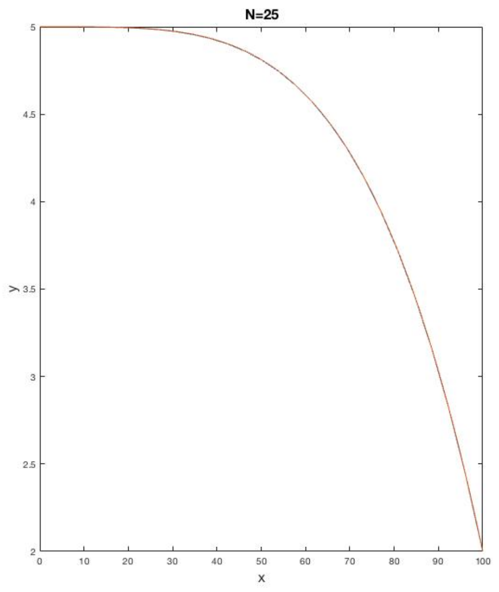


Figure 1: Error accumulation for different step size values, compared to its expected behaviour. The function $y'' = 12 \cdot \frac{\beta-\alpha}{L^4}x^2$ was evaluated on the interval [0,100], with the boundary values $\alpha = 5$ and $\beta = 2$.



(a) Approximation made with 5 evaluated points.



(b) Approximation made with 25 evaluated points.

Figure 2: The function $y'' = 12 \cdot \frac{\beta-\alpha}{L^4}x^2$ evaluated on the interval [0,100], with boundary values $\alpha = 5$ and $\beta = 2$. The blue graph show the approximated values and the red graph shows the exact solution.

2 Task 1.2

The beam function is presented in equation 2 [1]. u(x) describes the deflection of a beam with the load density q(x).

$$\begin{aligned} M''(x) &= q(x) \\ u''(x) &= \frac{M(x)}{EI} \end{aligned} \tag{2}$$

In plot 3, the solution $u(x)$ for the beam function is presented for a load density $q(x) = -50kN/m$, cross section moment of inertia $I(x) = 10^{-3}(3 - 2cos^{12}(\frac{\pi x}{L}))m^4$, elasticity module $E = 1,9 \cdot 10^{11}N/m^2$, beam length $L = 10$ and number of discretization points $N = 999$. This solution was derived by solving two boundary value problems with the *2pBVP* function created in task 1.1. $M''(x) = q(x)$ was solved whereafter the solution to this was used to solve $u''(x) = \frac{M(x)}{EI}$. The midpoint deflection was approximated to 11,741059 mm. The Matlab script is presented in Appendix, figure 14.

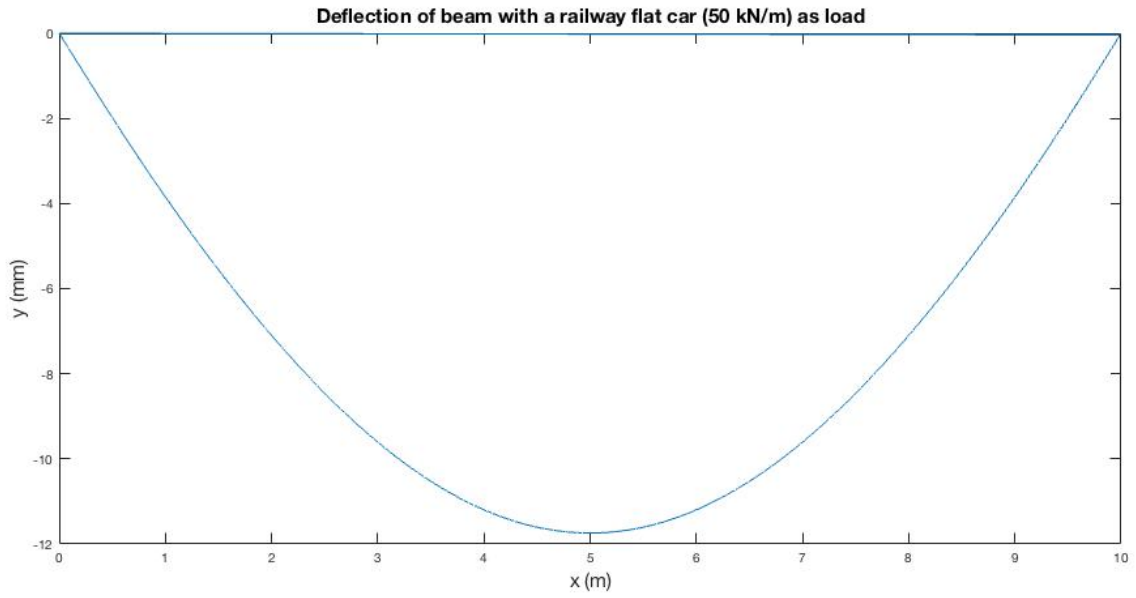


Figure 3: The deflection u presented in mm, as a function of x.

3 Task 2.1

The Sturm-Liouville eigenvalue problem is formulated in equation 3. To find its eigenvalues numerically the left hand side of the equation is discretized. Now approximations to the operator's eigenvalues are given by the eigenvalues of the discretization matrix, with the error $\mathcal{O}(\Delta x^2)$.

$$\frac{d}{dx}\left(p(x)\frac{d}{dx}\right) - q(x)y = \lambda u(x) \tag{3}$$

A Sturm-Liouville eigenvalue and eigenfunction solver *SturmLiouvilleSolve* was created in Matlab (see figure 15 in appendix for script). The discretization was done using a finite difference method of the second order. It was then tested on a simple

Sturm-Liouville problem with $p(x) = 1$ and $q(x) = 0$. This PDE is analytically solvable and therefore has known eigenvalues to which the approximations can be compared. The eigenvalues are presented in table 1.

Table 1: Approximated eigenvalues when N = 499 discrete function approximation points were used.

$\boldsymbol{\lambda_k}$	**Approximated $\boldsymbol{\lambda_k}$**	**Exact $\boldsymbol{\lambda_k}$= -(k$\boldsymbol{\pi}^2$)**	**Relative error**
λ_1	-9,86957193	-9,86960440	0,0003 %
λ_2	-39,47789809	-39,47841760	0,0013 %
λ_3	-88,82380960	-88,82643961	0,0030 %
λ_{22}	-4769,28715	-4776,88853	0,1591 %
λ_{100}	-95491,50281	-98696,04401	3,247 %
λ_{300}	-654508,4972	-888264,3961	26,31 %

The results in table 1 support that the *SturmLiouvilleSolve* function works as intended, since the first three approximated eigenvalues are close to the exact solutions. For these relative error is very small, less than 10^{-2}%. The approximated values are best for the first eigenvalue, after which the error increases. This is coherent with theory, which also states that the $\sqrt{N}$ first eigenvalues are good approximations (Source: Lecture Slides *Chapter3*). For N = 499 this means that the first 22 eigenvalues are good approximations. It is observed in table 1 that the 22^{nd} eigenvalue still has a relatively small deviation from the theoretical eigenvalue, but that it has grown a lot bigger compared to the first three eigenvalues. For the 100^{th} and 300^{th} eigenvalue the approximation is not good at all.

The error of the first three approximated eigenvalues is plotted as a function of step size in figure 4 (see script in figure 16 in appendix). As expected, the error is bigger for a bigger step size and goes to zero as the step size goes to zero. The error is also expected to have a square dependency of the step size since the approximation method used is of second order convergence. That this is the case can be seen by comparing the three measured slopes with the fourth function with the known slope of two in figure 4.

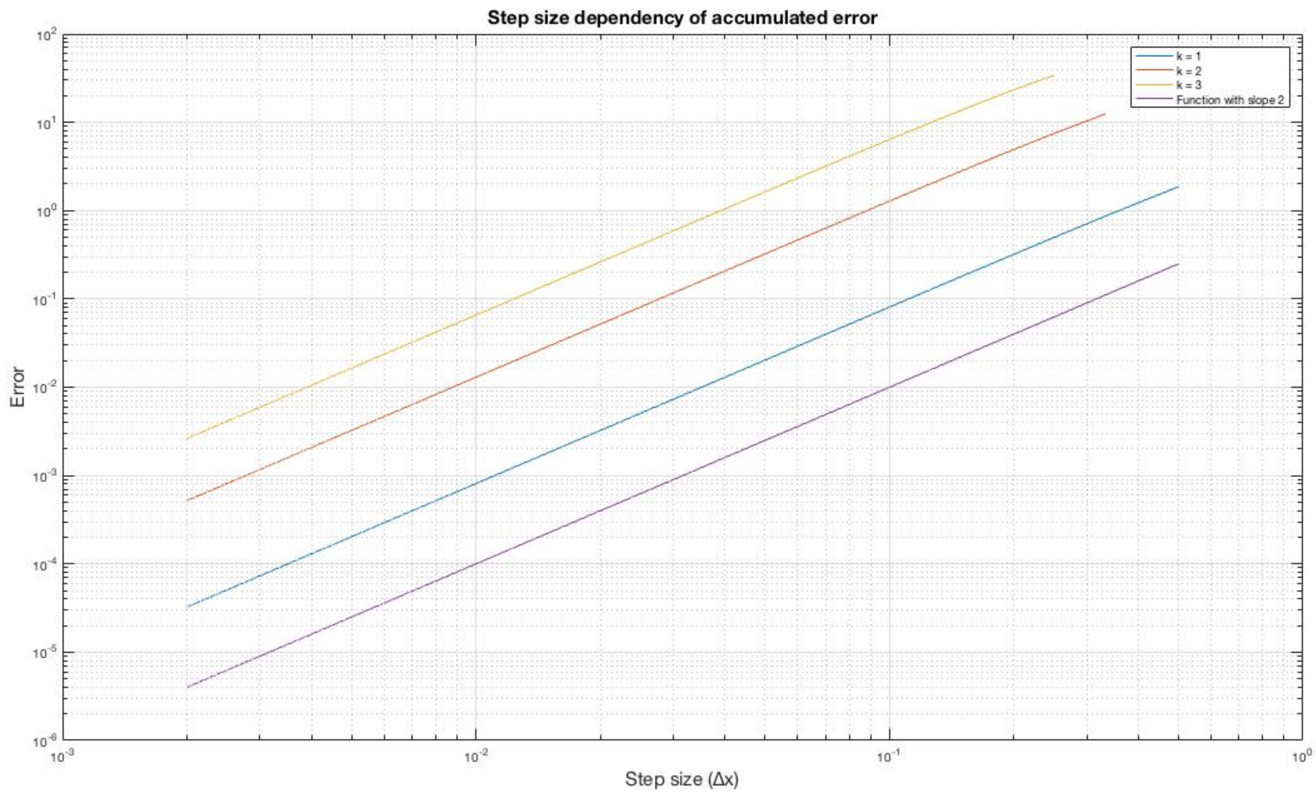


Figure 4: Error of eigenvalue approximations made using a second order finite difference method.

The first three eigenmodes calculated with *SturmLiouvilleSolve* are presented in figure 5. From analytical calculations it is known that the eigenfunctions should have a sinusoidal behaviour, but can be scaled with any scalar because of orthogonality. The eigenmodes returned by the function matched the calculated ones for the third one, but the first and second was scaled with a factor -1. Therefore the two first eigenmodes have been scaled to correspond to sinus functions with different frequency.

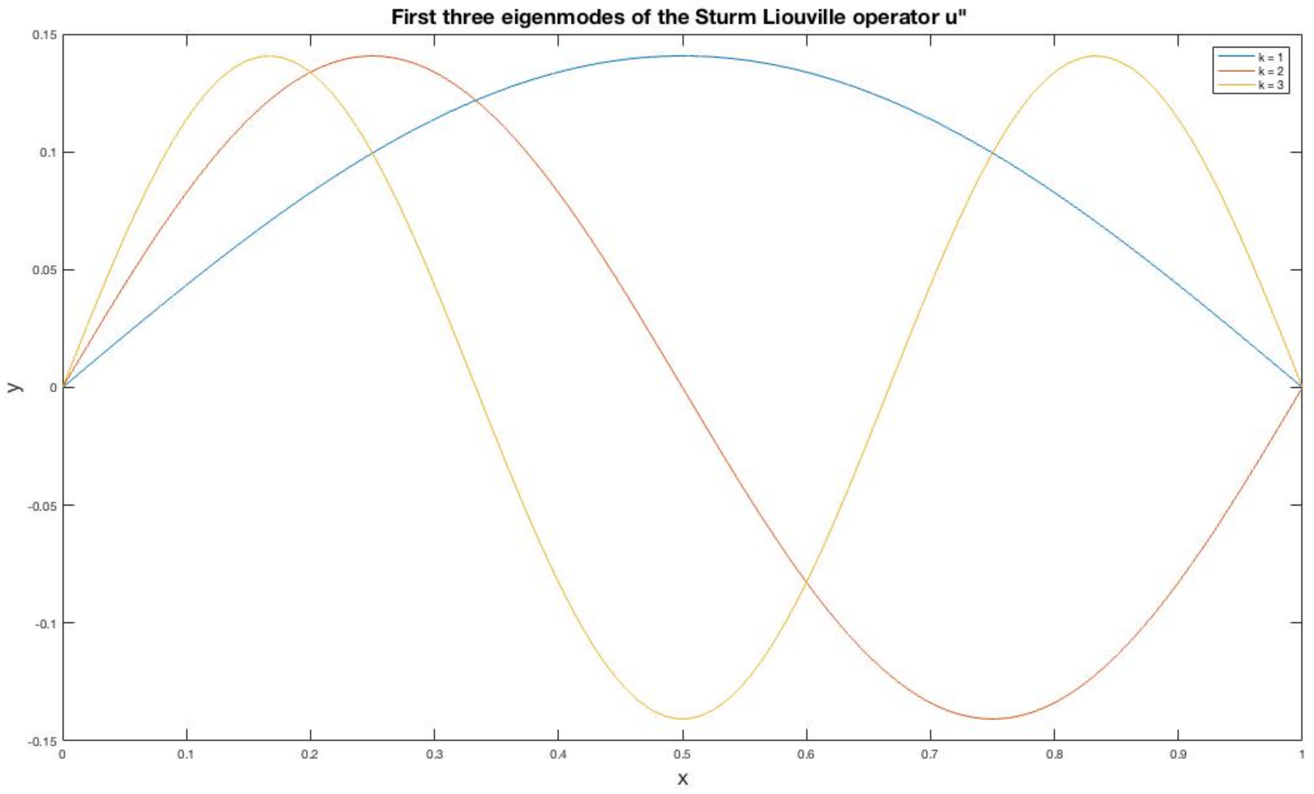


Figure 5: The first three eigenmodes of u".

4 Task 2.2

Equation 4 presents the Schrödinger equation, a PDE that can be treated as a Sturm-Liouville problem [1]. A Matlab function called sturm.m was created to solve its eigenvalues and eigenmodes, specifically the energies and states of the particle system it describes. The script for the solver is presented in Appendix, see figure 17.

$$\psi'' - V(x)\psi = -E\psi \tag{4}$$

In figure 6 the six, in magnitude smallest, eigenvectors are plotted as functions of x, without potential applied. To properly represent the difference in energy between these states, every wave function is lifted to its energy level, which is also its belonging eigenvalue, see equation 5. In this plot we see the characteristic sinus looking functions, where the length of the eigenfunctions increase with a half period for every k. The Matlab script which the wave functions where plotted in is presented in Appendix, see figure 18. This script used the Sturm-Liouville solver sturm.m.

$$|\Psi_k'(x)|^2 + E_k \tag{5}$$

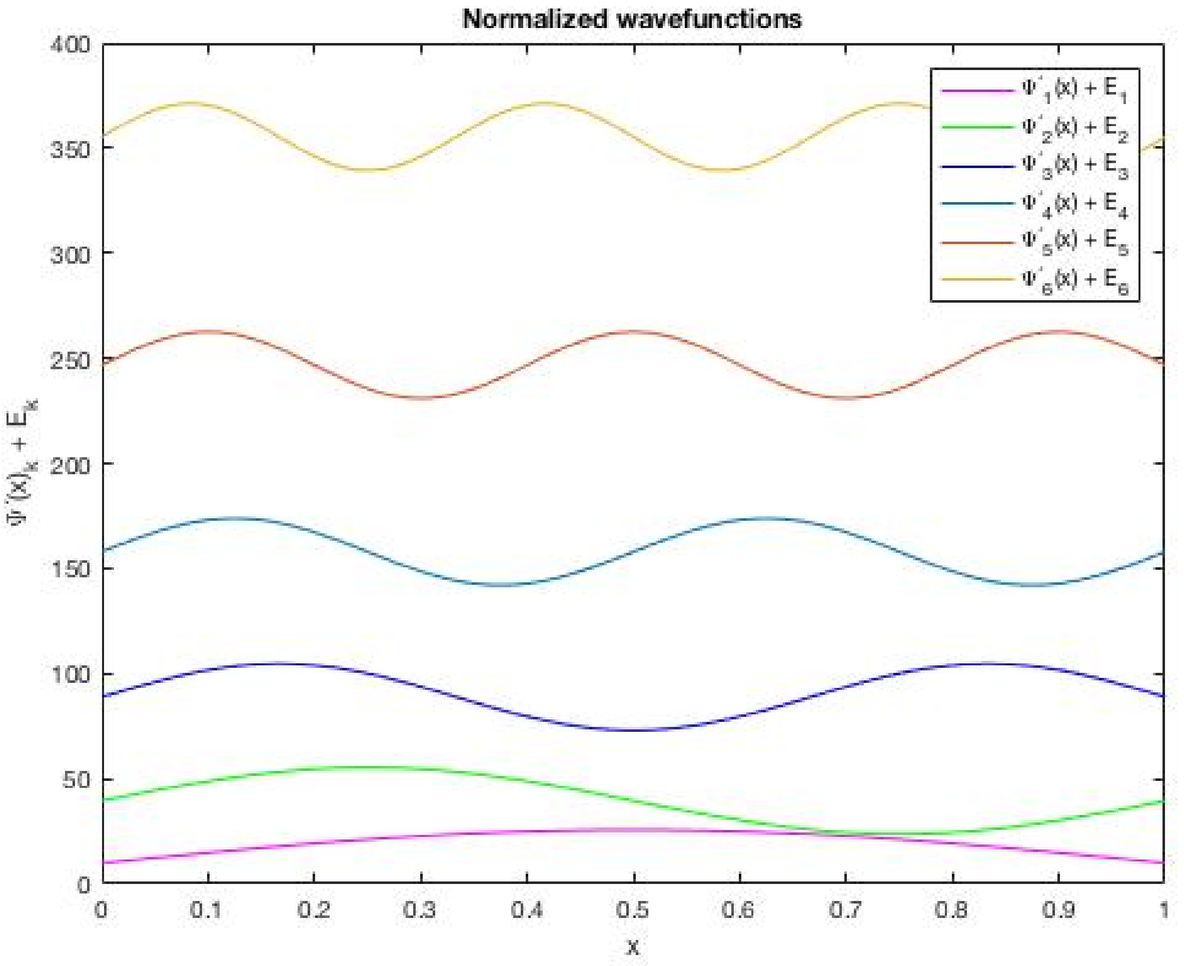


Figure 6: $\Psi'_k(x) + E_k$ with k = 1, 2, 3, 4, 5, 6 and V(x) = 0.

The probability density functions for the wave functions plotted in figure 6 above, are presented in the plot below, see figure 7. These are lifted to each energy level as well. Throughout whole task 2.2, $\Psi_k(x)$ and $|\Psi_k(x)|^2$ are normalized with a factor, to make them clearly presented.

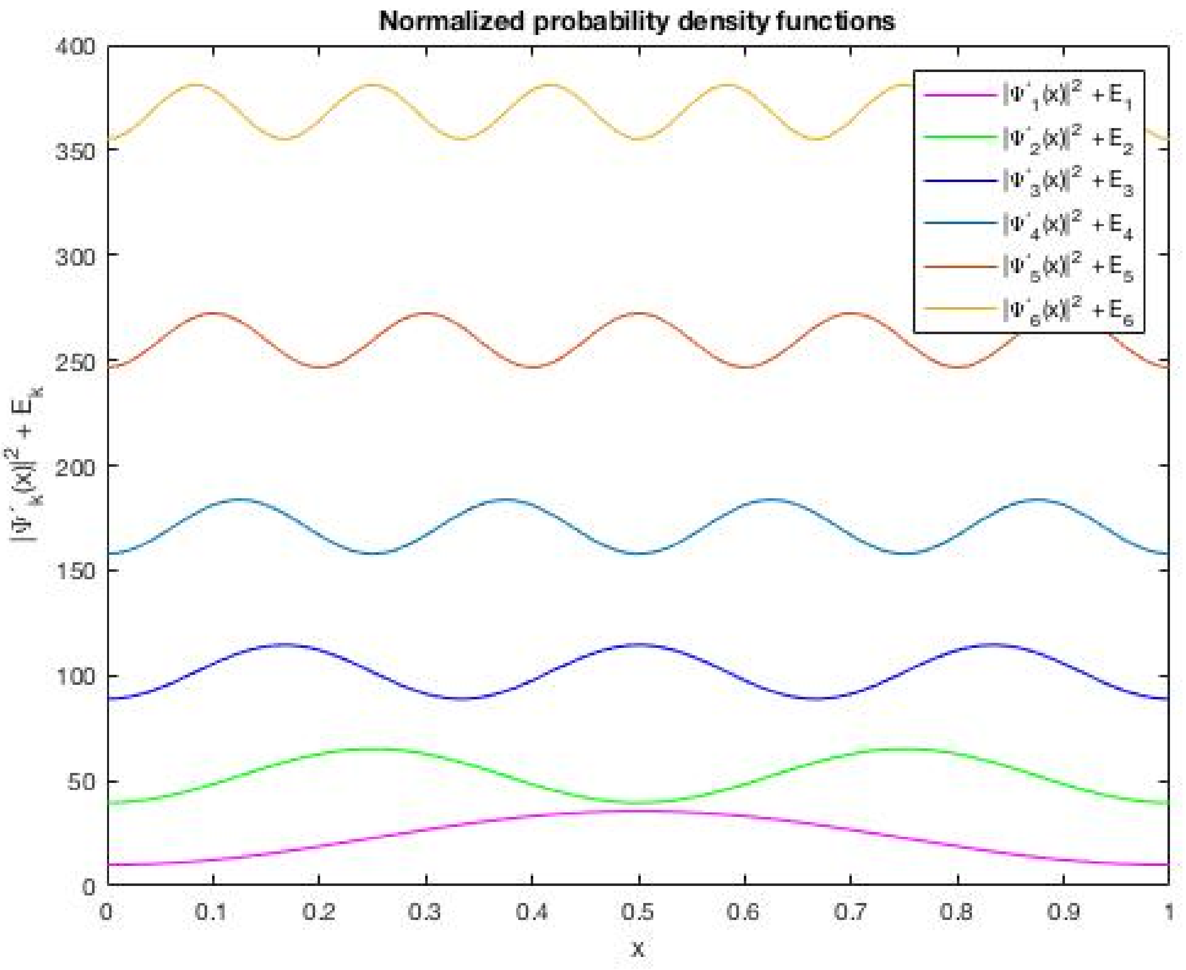


Figure 7: $|\Psi'_k(x)|^2 + E_k$ with k = 1, 2, 3, 4, 5, 6 and V(x) = 0.

Down below in figure 8 the wave functions with potential $V(x) = 700(0,5 - |0,5 - x|)$ applied, are presented. Their corresponding probability density functions are plotted in figure 9. The Matlab scripts are presented in Appendix, see figure 20 and figure 21. The plots are indicating on that the potential has its max value at x = 0,5, which can be confirmed by studying the potential function. The potential function should have a maximum value of $V(0,5) = 350$, which one can be convinced of by looking at figure 9. High potential results in a low probability, which especially the probability density functions for k = 1,2,3 and k = 4 visualize. The probability density functions for k = 5 and k = 6 seem to look almost like they do without applied potential (see figure 7) which can be explained by the maximum of the potential being below their energy levels.

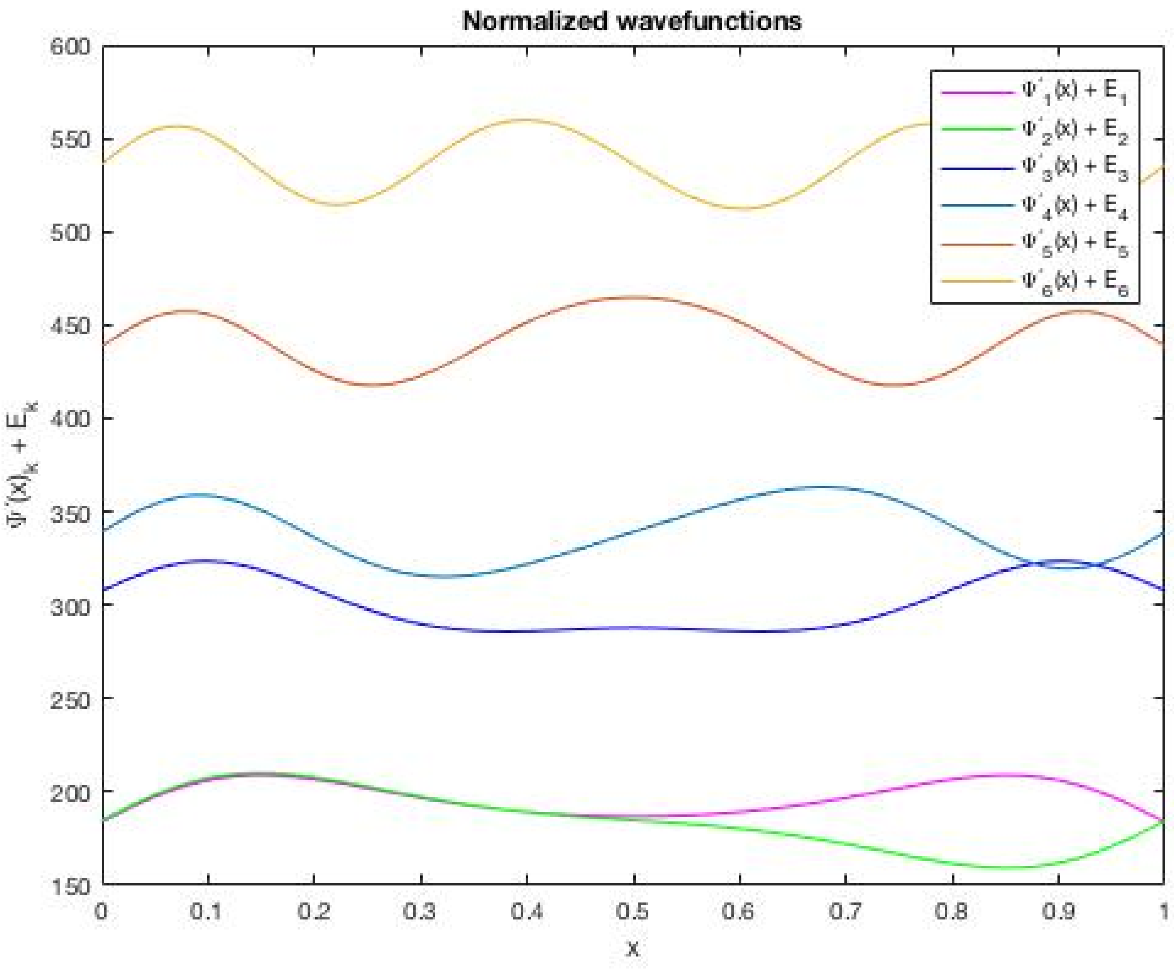


Figure 8: Normalized wave functions with $V(x) = 700(0,5 - |0,5 - x|)$.

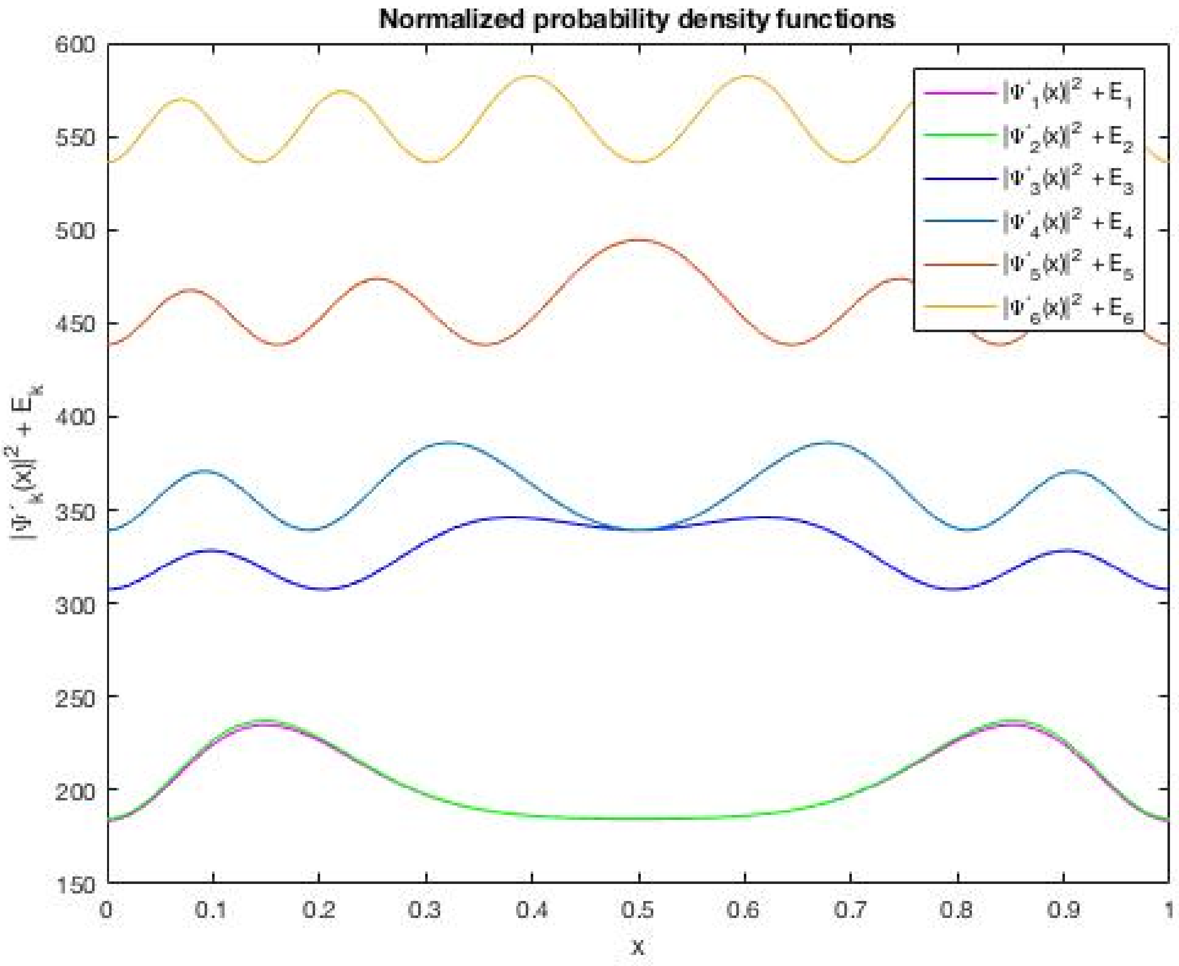


Figure 9: Normalized probability density functions with $V(x) = 700(0,5 - |0,5 - x|)$.

In figure 10 and figure 11 wave functions and probability density functions are plotted. The potential applied, $V(x) = \delta(x - 0,67) + \delta(x - 0,33)$, is supposed to represent potential walls at $x = 0,33$ and at $x = 0,67$. As shown in the figures, the wave functions and probability density functions behave as there are three wells instead of one. This is easily seen at especially the wave function with k = 1, as it has a maximum value and its characteristic form in each well. As well as the probability density functions, their belonging energy levels are plotted in figure 11. These represent where the wave function's probabilities are zero. This once again confirms that the two potential spikes divide the well into three wells, as the probability to find a particle at $x = 0,33$ and $x = 0,67$ is zero, or close to zero. The scripts for these plots are presented in Appendix, see figure 22 and figure 23.

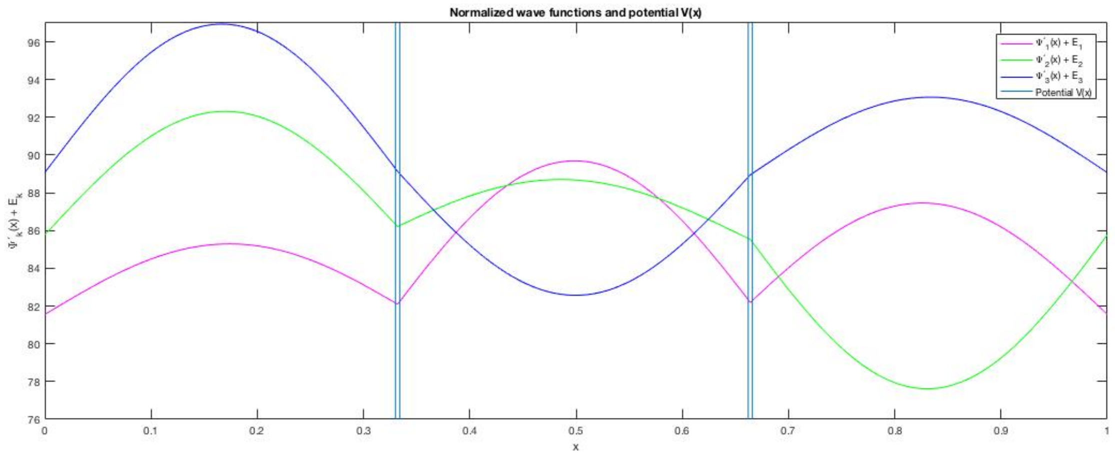


Figure 10: Normalized wave functions with $V(x) = \delta(x - 0,67) + \delta(x - 0,33)$.

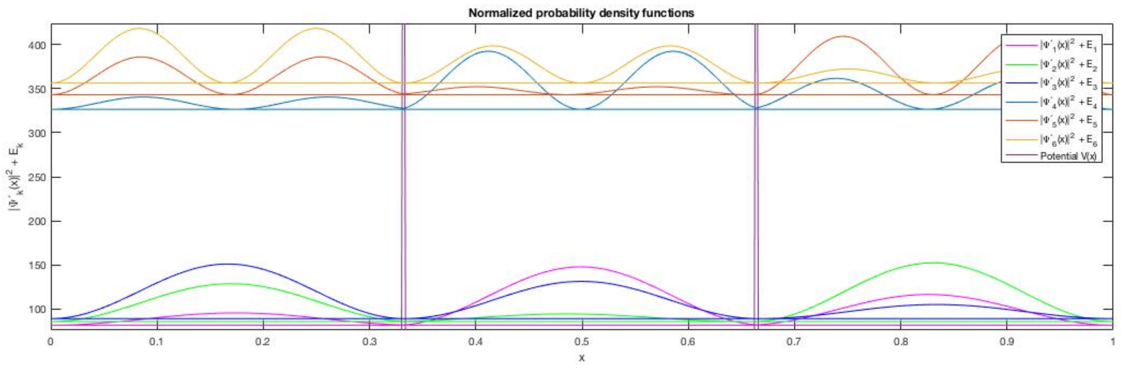


Figure 11: Normalized probability density functions with $V(x) = \delta(x-0,67)+\delta(x-0,33)$.

5 Acknowledgements

We want to thank our exercise mentor Peter Meisrimel, for explaining and helping with Matlab scripts. In addition we want to acknowledge our classmates Louise

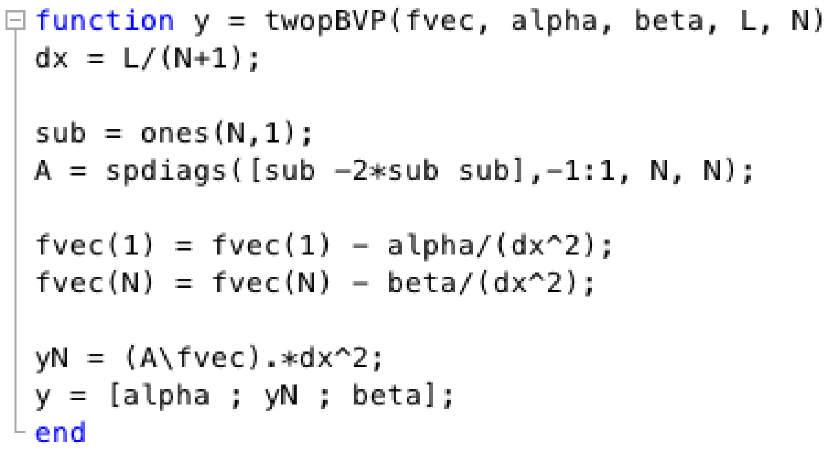
Karsten, Anna Sjerling, Isabelle Frodé and Sara Enander, for good discussions and sharing of knowledge.

References

[1] Stillfjord, T, Söderlind, G. *Project 2 in FMNN10 and NUMN12*. Lunds Tekniska Högskola. 2019.

6 Appendix - Matlab scripts

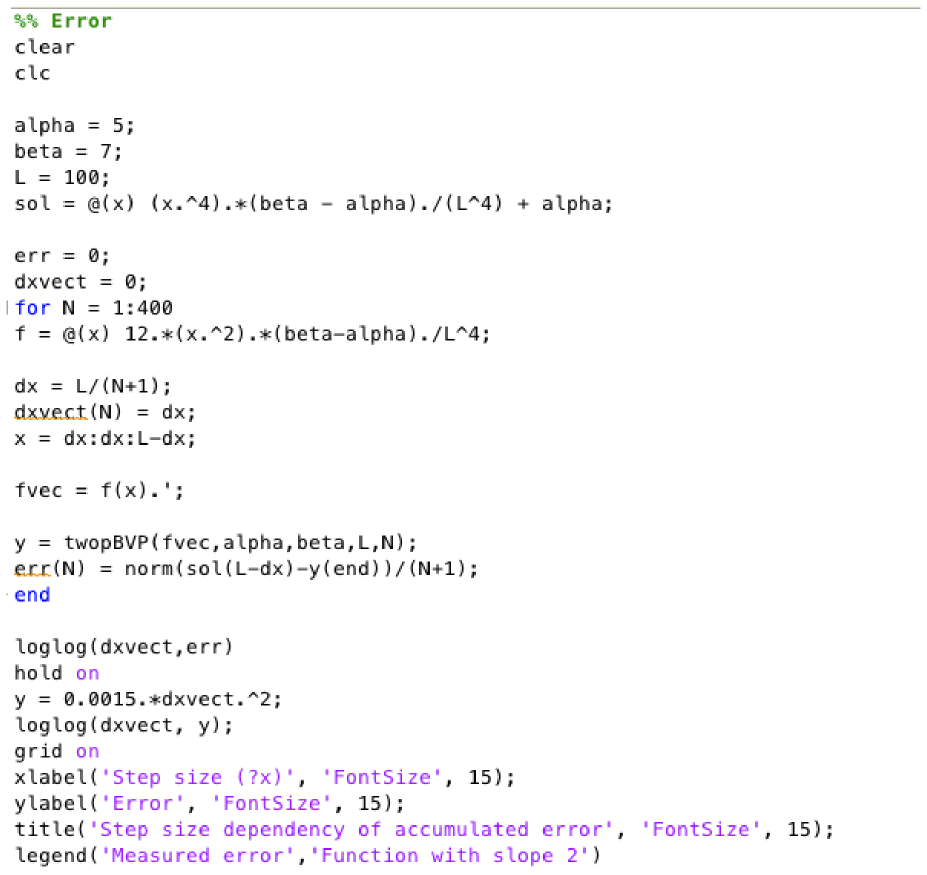
6.1 twopBVP.m



```
function y = twopBVP(fvec, alpha, beta, L, N)
dx = L/(N+1);

sub = ones(N,1);
A = spdiags([sub -2*sub sub],-1:1, N, N);

fvec(1) = fvec(1) - alpha/(dx^2);
fvec(N) = fvec(N) - beta/(dx^2);

yN = (A\fvec).*dx^2;
y = [alpha ; yN ; beta];
end
```

Figure 12: Script for the function twopBVP.

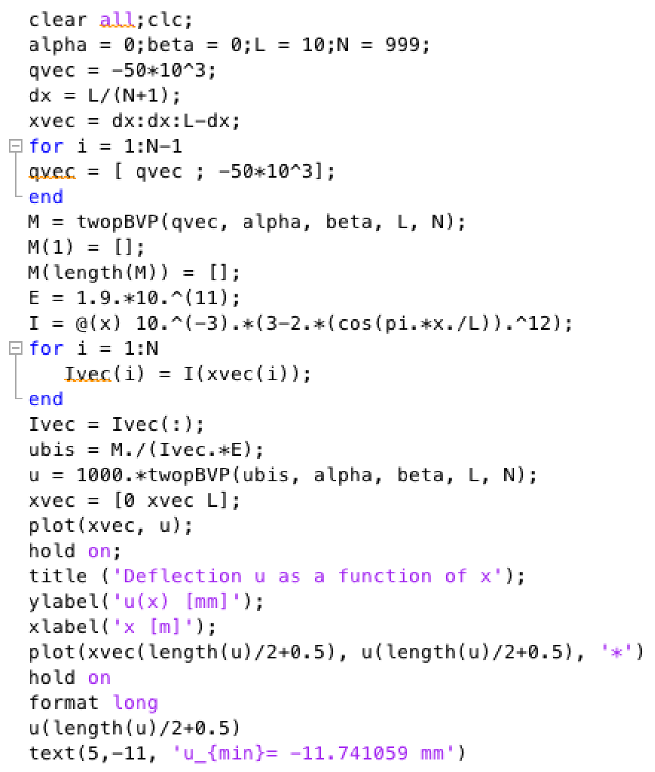
6.2 errorcalc



```
%% Error
clear
clc

alpha = 5;
beta = 7;
L = 100;
sol = @(x) (x.^4).*(beta - alpha)./(L^4) + alpha;

err = 0;
dxvect = 0;
for N = 1:400
f = @(x) 12.*(x.^2).*(beta-alpha)./L^4;

dx = L/(N+1);
dxvect(N) = dx;
x = dx:dx:L-dx;

fvec = f(x).';

y = twopBVP(fvec,alpha,beta,L,N);
err(N) = norm(sol(L-dx)-y(end))/(N+1);
end

loglog(dxvect,err)
hold on
y = 0.0015.*dxvect.^2;
loglog(dxvect, y);
grid on
xlabel('Step size (?x)', 'FontSize', 15);
ylabel('Error', 'FontSize', 15);
title('Step size dependency of accumulated error', 'FontSize', 15);
legend('Measured error','Function with slope 2')
```

Figure 13: Main script for calculating error of twopBVP.

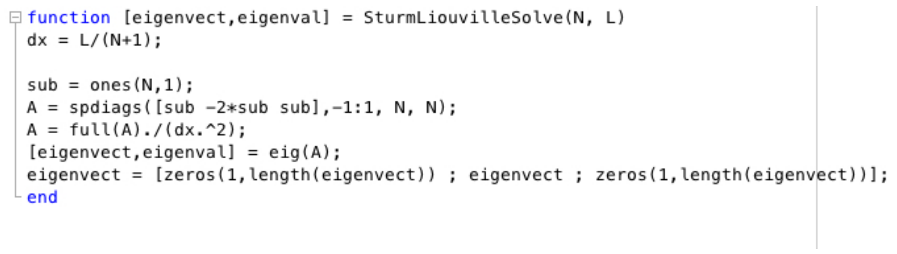
6.3 test12.m



```
clear all;clc;
alpha = 0;beta = 0;L = 10;N = 999;
qvec = -50*10^3;
dx = L/(N+1);
xvec = dx:dx:L-dx;
for i = 1:N-1
qvec = [ qvec ; -50*10^3];
end
M = twopBVP(qvec, alpha, beta, L, N);
M(1) = [];
M(length(M)) = [];
E = 1.9.*10.^(11);
I = @(x) 10.^(-3).*(3-2.*(cos(pi.*x./L)).^12);
for i = 1:N
   Ivec(i) = I(xvec(i));
end
Ivec = Ivec(:);
ubis = M./(Ivec.*E);
u = 1000.*twopBVP(ubis, alpha, beta, L, N);
xvec = [0 xvec L];
plot(xvec, u);
hold on;
title ('Deflection u as a function of x');
ylabel('u(x) [mm]');
xlabel('x [m]');
plot(xvec(length(u)/2+0.5), u(length(u)/2+0.5), '*')
hold on
format long
u(length(u)/2+0.5)
text(5,-11, 'u_{min}= -11.741059 mm')
```

Figure 14: Test script for task 2.1, solving the beam function.

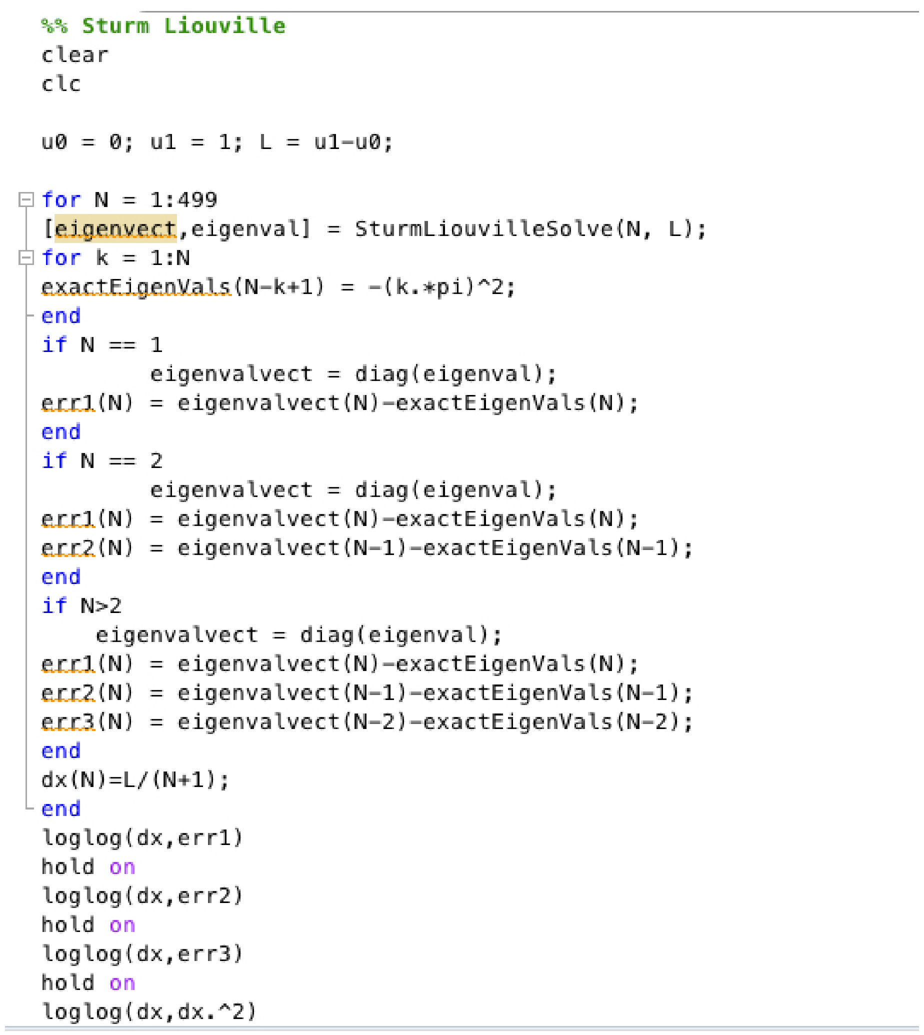
6.4 SturmLiouvilleSolve.m



```
function [eigenvect,eigenval] = SturmLiouvilleSolve(N, L)
dx = L/(N+1);

sub = ones(N,1);
A = spdiags([sub -2*sub sub],-1:1, N, N);
A = full(A)./(dx.^2);
[eigenvect,eigenval] = eig(A);
eigenvect = [zeros(1,length(eigenvect)) ; eigenvect ; zeros(1,length(eigenvect))];
end
```

Figure 15: Script for the function SturmLiouvilleSolve.

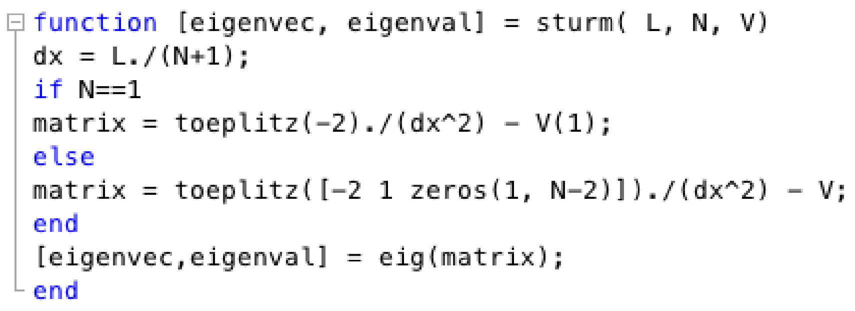
6.5 sturmerror.m



```
%% Sturm Liouville
clear
clc

u0 = 0; u1 = 1; L = u1-u0;

for N = 1:499
[eigenvect,eigenval] = SturmLiouvilleSolve(N, L);
for k = 1:N
exactEigenVals(N-k+1) = -(k.*pi)^2;
end
if N == 1
        eigenvalvect = diag(eigenval);
err1(N) = eigenvalvect(N)-exactEigenVals(N);
end
if N == 2
        eigenvalvect = diag(eigenval);
err1(N) = eigenvalvect(N)-exactEigenVals(N);
err2(N) = eigenvalvect(N-1)-exactEigenVals(N-1);
end
if N>2
    eigenvalvect = diag(eigenval);
err1(N) = eigenvalvect(N)-exactEigenVals(N);
err2(N) = eigenvalvect(N-1)-exactEigenVals(N-1);
err3(N) = eigenvalvect(N-2)-exactEigenVals(N-2);
end
dx(N)=L/(N+1);
end
loglog(dx,err1)
hold on
loglog(dx,err2)
hold on
loglog(dx,err3)
hold on
loglog(dx,dx.^2)
```

Figure 16: Script for calculating the error of the function SturmLiouvilleSolve.

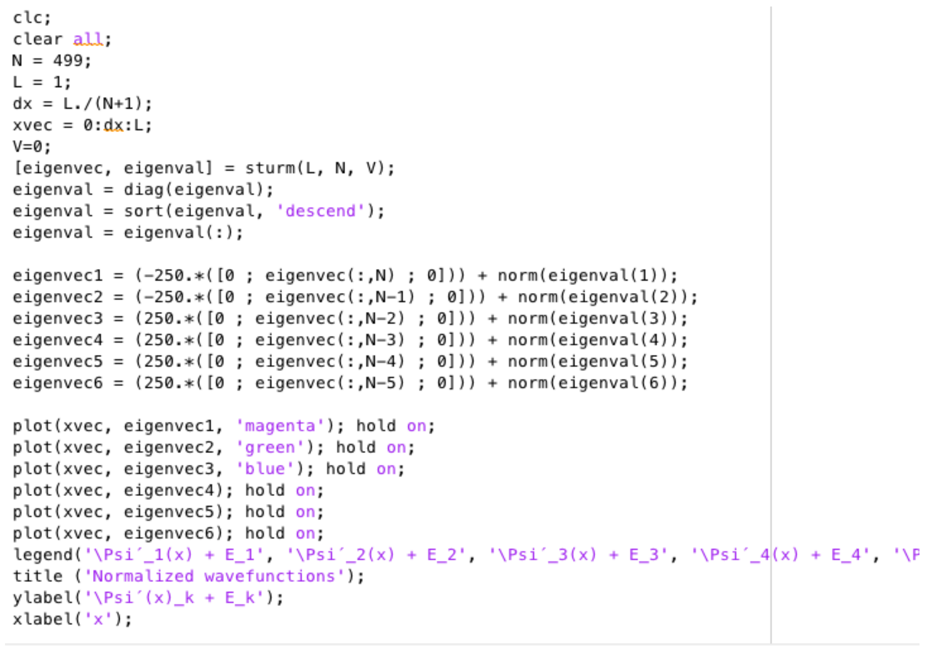
6.6 sturm.m



```
function [eigenvec, eigenval] = sturm( L, N, V)
dx = L./(N+1);
if N==1
matrix = toeplitz(-2)./(dx^2) - V(1);
else
matrix = toeplitz([-2 1 zeros(1, N-2)])./(dx^2) - V;
end
[eigenvec,eigenval] = eig(matrix);
end
```

Figure 17: Script for solving Sturm-Liouville problems.

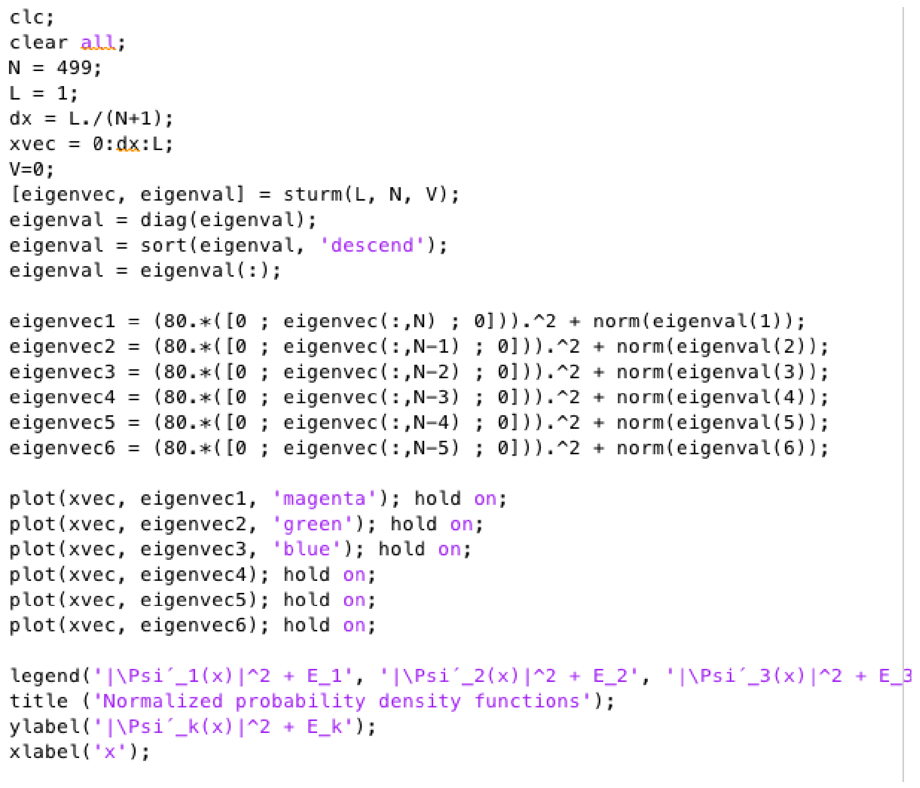
6.7 task22v0testwave.m



```
clc;
clear all;
N = 499;
L = 1;
dx = L./(N+1);
xvec = 0:dx:L;
V=0;
[eigenvec, eigenval] = sturm(L, N, V);
eigenval = diag(eigenval);
eigenval = sort(eigenval, 'descend');
eigenval = eigenval(:);

eigenvec1 = (-250.*([0 ; eigenvec(:,N) ; 0])) + norm(eigenval(1));
eigenvec2 = (-250.*([0 ; eigenvec(:,N-1) ; 0])) + norm(eigenval(2));
eigenvec3 = (250.*([0 ; eigenvec(:,N-2) ; 0])) + norm(eigenval(3));
eigenvec4 = (250.*([0 ; eigenvec(:,N-3) ; 0])) + norm(eigenval(4));
eigenvec5 = (250.*([0 ; eigenvec(:,N-4) ; 0])) + norm(eigenval(5));
eigenvec6 = (250.*([0 ; eigenvec(:,N-5) ; 0])) + norm(eigenval(6));

plot(xvec, eigenvec1, 'magenta'); hold on;
plot(xvec, eigenvec2, 'green'); hold on;
plot(xvec, eigenvec3, 'blue'); hold on;
plot(xvec, eigenvec4); hold on;
plot(xvec, eigenvec5); hold on;
plot(xvec, eigenvec6); hold on;
legend('\Psi´_1(x) + E_1', '\Psi´_2(x) + E_2', '\Psi´_3(x) + E_3', '\Psi´_4(x) + E_4', '\P
title ('Normalized wavefunctions');
ylabel('\Psi´(x)_k + E_k');
xlabel('x');
```

Figure 18: Script for plotting wave functions with V(x) = 0.

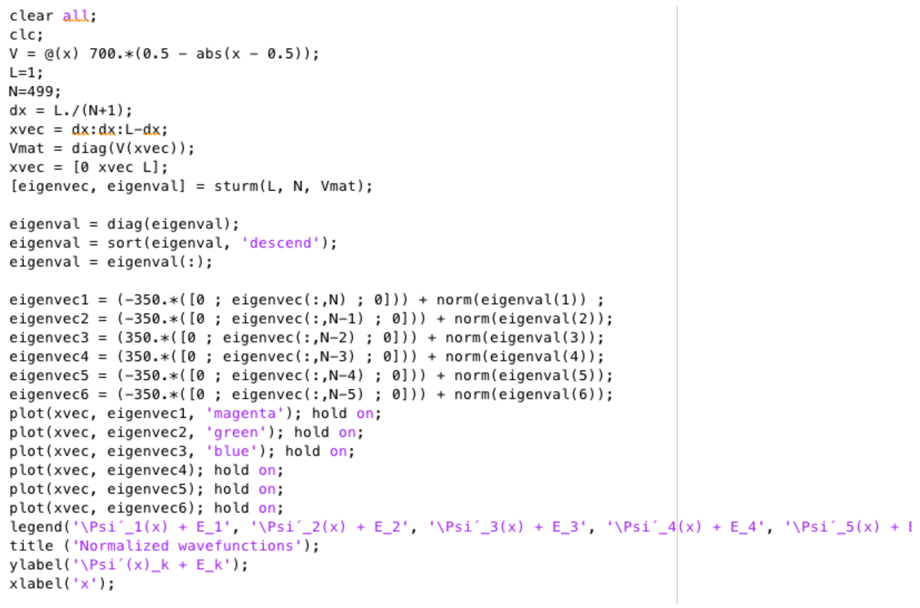
6.8 task22v0testprob.m



```
clc;
clear all;
N = 499;
L = 1;
dx = L./(N+1);
xvec = 0:dx:L;
V=0;
[eigenvec, eigenval] = sturm(L, N, V);
eigenval = diag(eigenval);
eigenval = sort(eigenval, 'descend');
eigenval = eigenval(:);

eigenvec1 = (80.*([0 ; eigenvec(:,N) ; 0])).^2 + norm(eigenval(1));
eigenvec2 = (80.*([0 ; eigenvec(:,N-1) ; 0])).^2 + norm(eigenval(2));
eigenvec3 = (80.*([0 ; eigenvec(:,N-2) ; 0])).^2 + norm(eigenval(3));
eigenvec4 = (80.*([0 ; eigenvec(:,N-3) ; 0])).^2 + norm(eigenval(4));
eigenvec5 = (80.*([0 ; eigenvec(:,N-4) ; 0])).^2 + norm(eigenval(5));
eigenvec6 = (80.*([0 ; eigenvec(:,N-5) ; 0])).^2 + norm(eigenval(6));

plot(xvec, eigenvec1, 'magenta'); hold on;
plot(xvec, eigenvec2, 'green'); hold on;
plot(xvec, eigenvec3, 'blue'); hold on;
plot(xvec, eigenvec4); hold on;
plot(xvec, eigenvec5); hold on;
plot(xvec, eigenvec6); hold on;

legend('|\Psi´_1(x)|^2 + E_1', '|\Psi´_2(x)|^2 + E_2', '|\Psi´_3(x)|^2 + E_3
title ('Normalized probability density functions');
ylabel('|\Psi´_k(x)|^2 + E_k');
xlabel('x');
```

Figure 19: Script for plotting probability density functions with V(x) = 0.

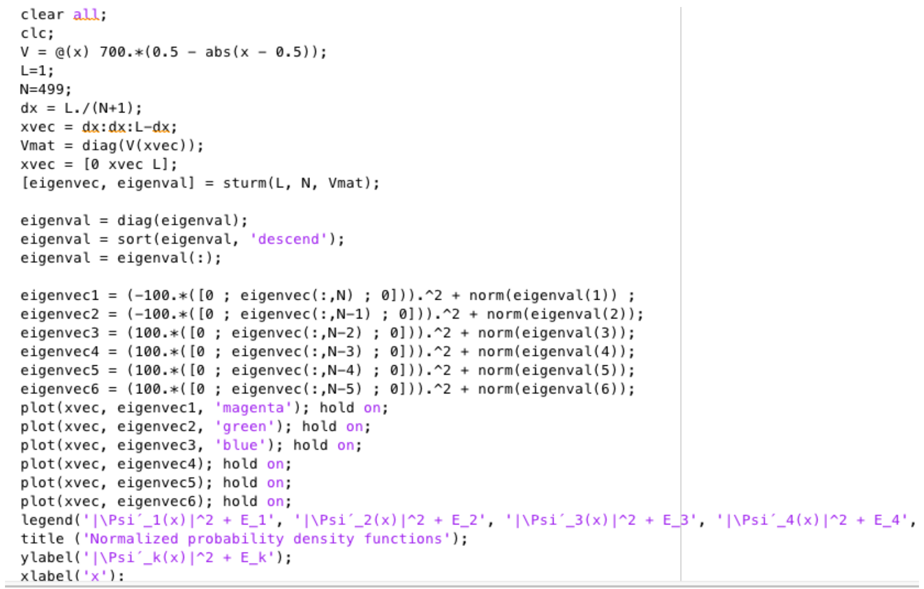
6.9 task22v1testwave.m



```
clear all;
clc;
V = @(x) 700.*(0.5 - abs(x - 0.5));
L=1;
N=499;
dx = L./(N+1);
xvec = dx:dx:L-dx;
Vmat = diag(V(xvec));
xvec = [0 xvec L];
[eigenvec, eigenval] = sturm(L, N, Vmat);

eigenval = diag(eigenval);
eigenval = sort(eigenval, 'descend');
eigenval = eigenval(:);

eigenvec1 = (-350.*([0 ; eigenvec(:,N) ; 0])) + norm(eigenval(1)) ;
eigenvec2 = (-350.*([0 ; eigenvec(:,N-1) ; 0])) + norm(eigenval(2));
eigenvec3 = (350.*([0 ; eigenvec(:,N-2) ; 0])) + norm(eigenval(3));
eigenvec4 = (350.*([0 ; eigenvec(:,N-3) ; 0])) + norm(eigenval(4));
eigenvec5 = (-350.*([0 ; eigenvec(:,N-4) ; 0])) + norm(eigenval(5));
eigenvec6 = (-350.*([0 ; eigenvec(:,N-5) ; 0])) + norm(eigenval(6));
plot(xvec, eigenvec1, 'magenta'); hold on;
plot(xvec, eigenvec2, 'green'); hold on;
plot(xvec, eigenvec3, 'blue'); hold on;
plot(xvec, eigenvec4); hold on;
plot(xvec, eigenvec5); hold on;
plot(xvec, eigenvec6); hold on;
legend('\Psi´_1(x) + E_1', '\Psi´_2(x) + E_2', '\Psi´_3(x) + E_3', '\Psi´_4(x) + E_4', '\Psi´_5(x) +
title ('Normalized wavefunctions');
ylabel('\Psi´(x)_k + E_k');
xlabel('x');
```

Figure 20: Script for plotting wave functions with $V(x) = 700(0,5 - |0,5 - x|)$.

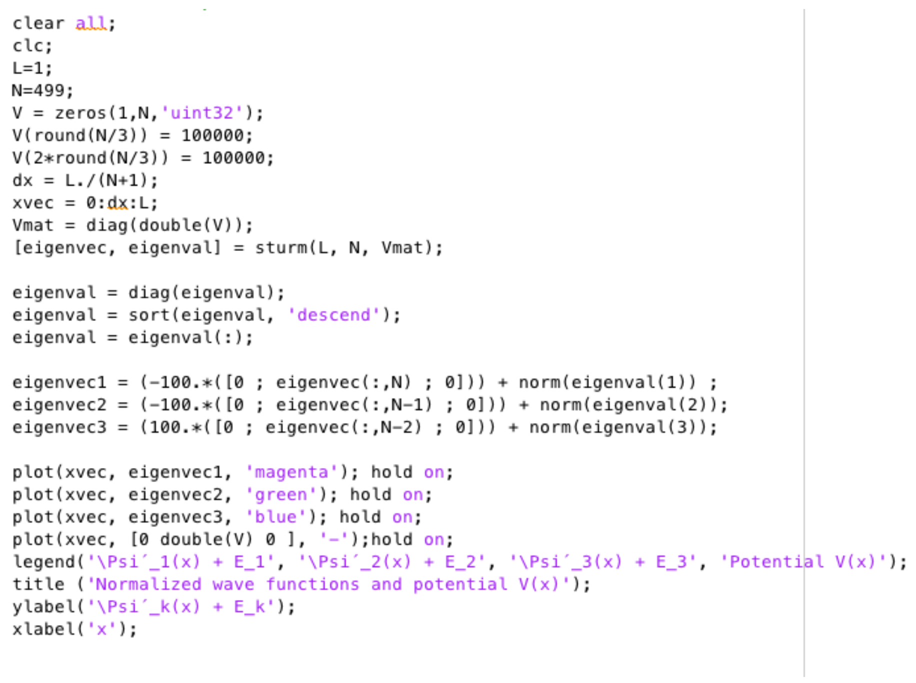
6.10 task22v1testprob.m



```
clear all;
clc;
V = @(x) 700.*(0.5 - abs(x - 0.5));
L=1;
N=499;
dx = L./(N+1);
xvec = dx:dx:L-dx;
Vmat = diag(V(xvec));
xvec = [0 xvec L];
[eigenvec, eigenval] = sturm(L, N, Vmat);

eigenval = diag(eigenval);
eigenval = sort(eigenval, 'descend');
eigenval = eigenval(:);

eigenvec1 = (-100.*([0 ; eigenvec(:,N) ; 0])).^2 + norm(eigenval(1)) ;
eigenvec2 = (-100.*([0 ; eigenvec(:,N-1) ; 0])).^2 + norm(eigenval(2));
eigenvec3 = (100.*([0 ; eigenvec(:,N-2) ; 0])).^2 + norm(eigenval(3));
eigenvec4 = (100.*([0 ; eigenvec(:,N-3) ; 0])).^2 + norm(eigenval(4));
eigenvec5 = (100.*([0 ; eigenvec(:,N-4) ; 0])).^2 + norm(eigenval(5));
eigenvec6 = (100.*([0 ; eigenvec(:,N-5) ; 0])).^2 + norm(eigenval(6));
plot(xvec, eigenvec1, 'magenta'); hold on;
plot(xvec, eigenvec2, 'green'); hold on;
plot(xvec, eigenvec3, 'blue'); hold on;
plot(xvec, eigenvec4); hold on;
plot(xvec, eigenvec5); hold on;
plot(xvec, eigenvec6); hold on;
legend('|\Psi´_1(x)|^2 + E_1', '|\Psi´_2(x)|^2 + E_2', '|\Psi´_3(x)|^2 + E_3', '|\Psi´_4(x)|^2 + E_4',
title ('Normalized probability density functions');
ylabel('|\Psi´_k(x)|^2 + E_k');
xlabel('x');
```

Figure 21: Script for plotting probability density functions with $V(x) = 700(0,5 - |0,5 - x|)$.

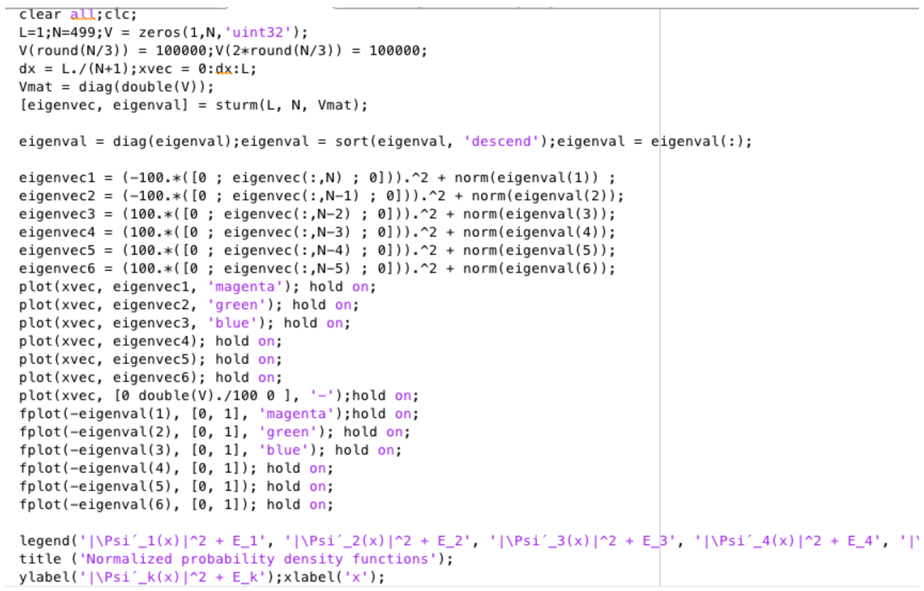
6.11 task22diractestwave.m



```
clear all;
clc;
L=1;
N=499;
V = zeros(1,N,'uint32');
V(round(N/3)) = 100000;
V(2*round(N/3)) = 100000;
dx = L./(N+1);
xvec = 0:dx:L;
Vmat = diag(double(V));
[eigenvec, eigenval] = sturm(L, N, Vmat);

eigenval = diag(eigenval);
eigenval = sort(eigenval, 'descend');
eigenval = eigenval(:);

eigenvec1 = (-100.*([0 ; eigenvec(:,N) ; 0])) + norm(eigenval(1)) ;
eigenvec2 = (-100.*([0 ; eigenvec(:,N-1) ; 0])) + norm(eigenval(2));
eigenvec3 = (100.*([0 ; eigenvec(:,N-2) ; 0])) + norm(eigenval(3));

plot(xvec, eigenvec1, 'magenta'); hold on;
plot(xvec, eigenvec2, 'green'); hold on;
plot(xvec, eigenvec3, 'blue'); hold on;
plot(xvec, [0 double(V) 0 ], '-');hold on;
legend('\Psi´_1(x) + E_1', '\Psi´_2(x) + E_2', '\Psi´_3(x) + E_3', 'Potential V(x)');
title ('Normalized wave functions and potential V(x)');
ylabel('\Psi´_k(x) + E_k');
xlabel('x');
```

Figure 22: Script for plotting wave functions with $V(x) = \delta(x - 0,67) + \delta(x - 0,33)$.

6.12 task22diractestprob.m



```
clear all;clc;
L=1;N=499;V = zeros(1,N,'uint32');
V(round(N/3)) = 100000;V(2*round(N/3)) = 100000;
dx = L./(N+1);xvec = 0:dx:L;
Vmat = diag(double(V));
[eigenvec, eigenval] = sturm(L, N, Vmat);

eigenval = diag(eigenval);eigenval = sort(eigenval, 'descend');eigenval = eigenval(:);

eigenvec1 = (-100.*([0 ; eigenvec(:,N) ; 0])).^2 + norm(eigenval(1)) ;
eigenvec2 = (-100.*([0 ; eigenvec(:,N-1) ; 0])).^2 + norm(eigenval(2));
eigenvec3 = (100.*([0 ; eigenvec(:,N-2) ; 0])).^2 + norm(eigenval(3));
eigenvec4 = (100.*([0 ; eigenvec(:,N-3) ; 0])).^2 + norm(eigenval(4));
eigenvec5 = (100.*([0 ; eigenvec(:,N-4) ; 0])).^2 + norm(eigenval(5));
eigenvec6 = (100.*([0 ; eigenvec(:,N-5) ; 0])).^2 + norm(eigenval(6));
plot(xvec, eigenvec1, 'magenta'); hold on;
plot(xvec, eigenvec2, 'green'); hold on;
plot(xvec, eigenvec3, 'blue'); hold on;
plot(xvec, eigenvec4); hold on;
plot(xvec, eigenvec5); hold on;
plot(xvec, eigenvec6); hold on;
plot(xvec, [0 double(V)./100 0 ], '-');hold on;
fplot(-eigenval(1), [0, 1], 'magenta');hold on;
fplot(-eigenval(2), [0, 1], 'green'); hold on;
fplot(-eigenval(3), [0, 1], 'blue'); hold on;
fplot(-eigenval(4), [0, 1]); hold on;
fplot(-eigenval(5), [0, 1]); hold on;
fplot(-eigenval(6), [0, 1]); hold on;

legend('|\Psi´_1(x)|^2 + E_1', '|\Psi´_2(x)|^2 + E_2', '|\Psi´_3(x)|^2 + E_3', '|\Psi´_4(x)|^2 + E_4', '|
title ('Normalized probability density functions');
ylabel('|\Psi´_k(x)|^2 + E_k');xlabel('x');
```

Figure 23: Script for plotting probability density functions with $V(x) = \delta(x-0,67)+\delta(x-0,33)$.