Scientific Computing - A.Y. 2022/2023

Project No. 1

Vincenzo Antonio Isoldi
Vittorio Mocchi

March 31, 2023

Exercise 1

Introduction

This exercise aims to compute and analyze two-dimensional *random walks* on a lattice.
The report provides an executive summary of the major findings, including computational and theoretical analysis of the mathematical problem. We will start by defining what a *random walk* is and then use the numeric computing environment MATLAB to implement and visualize it.

Definitions

Random Walk A *random walk* is a random process that describes a path that consists of a succession of random steps on some mathematical space.

Random Walk on a Lattice A popular random walk model is that of a random walk on a regular lattice, where at each step the location jumps to another site according to some probability distribution. In a *simple random walk*, the location can only jump to neighboring sites of the lattice, forming a lattice path. In a *simple symmetric random walk* on a locally finite lattice, the probabilities of the location jumping to each one of its immediate neighbors are the same. From now on, we will assume our 2-D lattice to be $\mathbb{Z}^2$.

Analysis

Code summary We first use the MATLAB script `runRW2D.m` to generate a 2-D *random walk* with N steps plus the initial position. To do so, we use the function `randomwalk2D.m` which represents the core of our computation. This function takes two arguments: N, the number of steps we want to compute, and `toplot`, a string that indicates whether to plot the generated *random walk* or not. The initial position is first initialized to $(0, 0)$ and then - through a combination of the MATLAB functions `rem` and `randi` - we generate N independent movements on the x-axis. We collect the independent movements on a vector, `xDirection`, and then we compute the N movements on the `y-axis`, collecting them on another vector, `yDirection`. Let's observe that the components of the `yDirection` depend on those on the `xDirection`. The movements in the x and y directions are then combined to create a single vector of $N + 1$ positions and finally, using the `cumsum` function, we obtain the vector `P`, that takes track of the coordinates evolution of the *random walk* after each step. If *toplot* is set to 'true', the resulting path is plotted.

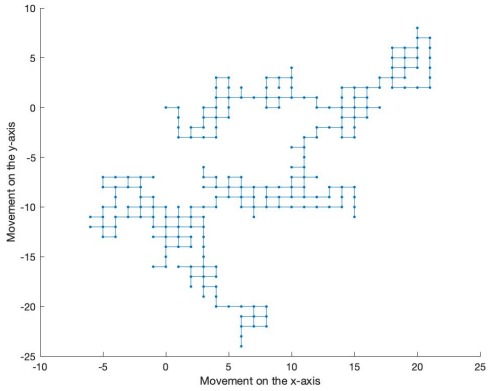

Figure 1: A symmetrical 2-D random walk with N=500 steps

Finally, we can compute the variance of the final position of the particle on each axis for different values of N.

Mathematical modeling We now want to verify that the generated *random walk* is symmetric, as requested. Let's consider two different real random variables, X and Y, and the random vector (X, Y). At each step, our point can go:

- North (N), which corresponds to $(X, Y) = (0, 1)$;
- South (S), which corresponds to $(X, Y) = (0, -1)$;
- East (E), which corresponds to $(X, Y) = (1, 0)$;
- West (W), which corresponds to $(X, Y) = (-1, 0)$.

We want:

$$P((X,Y) = (0,1)) = P((X,Y) = (0,-1)) = P((X,Y) = (1,0)) = P((X,Y) = (-1,0)) = \frac{1}{4}$$

Going back to the MATLAB script `randomwalk2D.m` at rows `4-6`, we have:

```
P=[0,0]; % Initial position.
xDirection=rem(randi([-1 2],N,1),2);
```

With `randi([-1 2],N,1)` we generate N independent numbers $(-1, 0, +1, 2)$. Applying the `rem` function to the obtained vector, we identify 2 with 0 reminder, obtaining N independent movements on the x-axis, $(-1, 0, 1)$, with probabilities $P(X = 1) = P(X = -1) = \frac{1}{4}$ and $P(X = 0) = \frac{1}{2}$.
Now we want to assign the values of Y, conditional upon the values of X.
If $X = -1$ or $X = 1$, Y must take the value 0, since the only two allowed movements with $X \in \{1, -1\}$ are $(X, Y) \in \{(1, 0), (-1, 0)\}$ corresponding to E and W.
Thus, we have $P(Y = 0|X = 1) = P(Y = 0|X = -1) = 1$ and from the *conditional probability* formula, we obtain:

- $P((X,Y) = (1,0)) = P(Y = 0|X = 1) \cdot P(X = 1) = 1 \cdot \frac{1}{4} = \frac{1}{4}$
- $P((X,Y) = (-1,0)) = P(Y = 0|X = -1) \cdot P(X = -1) = 1 \cdot \frac{1}{4} = \frac{1}{4}$

If $X = 0$, Y can assume both the values -1 or 1, since the two allowed movements with $X = 0$ are $(X, Y) \in \{(0, 1), (0, -1)\}$ corresponding to N and S.
Thus, we have $P(Y = 0|X = 1) = P(Y = 0|X = -1) = \frac{1}{2}$ and from the *conditional probability* formula, we now obtain:

- $P((X,Y) = (0,1)) = P(Y = 1|X = 0) \cdot P(X = 0) = \frac{1}{2} \cdot \ \frac{1}{2} = \frac{1}{4}$
- $P((X,Y) = (0,-1)) = P(Y = -1|X = 0) \cdot P(X = 0) = \frac{1}{2} \cdot \ \frac{1}{2} = \frac{1}{4}$

which is exactly what we wanted.
The following MATLAB code from `randomwalk2D.m`, rows `4-6`, shows an implementation of the above argument:

```
for i=1:N
    if xDirection(i)==1 || xDirection(i)==-1
        yDirection(i)=0;
    else
```

```
            yDirection(i)=2*randi([0 1])-1;
        end
    end
```

If we want an empirical proof of the accuracy of our code, we can generate a large amount of *random walks* with a considerable number of steps N.

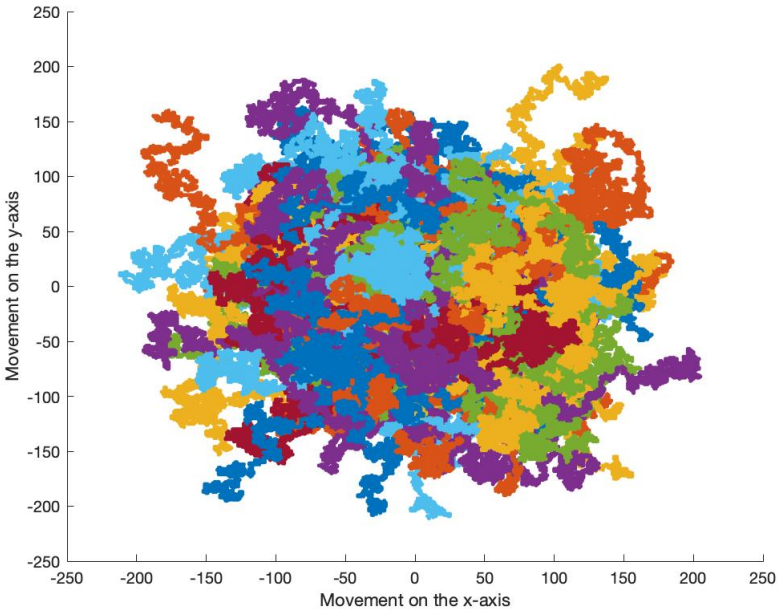

Figure 2: 300 random walks with 10000 steps

As we can see in Figure 2, we obtain a pretty neat rhomboidal figure centered in $(0, 0)$ and equally scaled among the axes. This suggests that the 2-D *random walk* is symmetric on the lattice, thus the algorithm used to compute our probabilities must be correct.
To find the average distance traveled in each direction, we can use the *expected value* (first moment) and the *variance* (second moment) of the final position of the walkers after N steps.
In particular, since each walker takes on average $N/2$ steps along each coordinate direction, the final position of a walker follows a *normal distribution* with *mean* 0 and *variance* $N/2$ along each coordinate.
Thus, the average distance traveled along each coordinate is the square root of the variance, which is equal to $\sqrt{N/2}$. Also, the total average distance traveled in 2-D is the square root of the sum of the variances along each coordinate, i.e. $\sqrt{2 \cdot \ N/2} = \sqrt{N}$.
To verify this result experimentally, we can run the second part of the MATLAB script `runRW2D.m` to simulate 1000 walkers starting from the origin in a two-dimensional symmetrical lattice and let them evolve for a number of steps $N = (10\ 20\ 40\ 60\ 160\ 320\ 640)$. We can then calculate the final position of each walker and the average distance traveled along each coordinate by computing the variance of the final position in each of the axes over every trial, with a fixed steps number.
After plotting our results, we obtain the graph found in Figure 3.

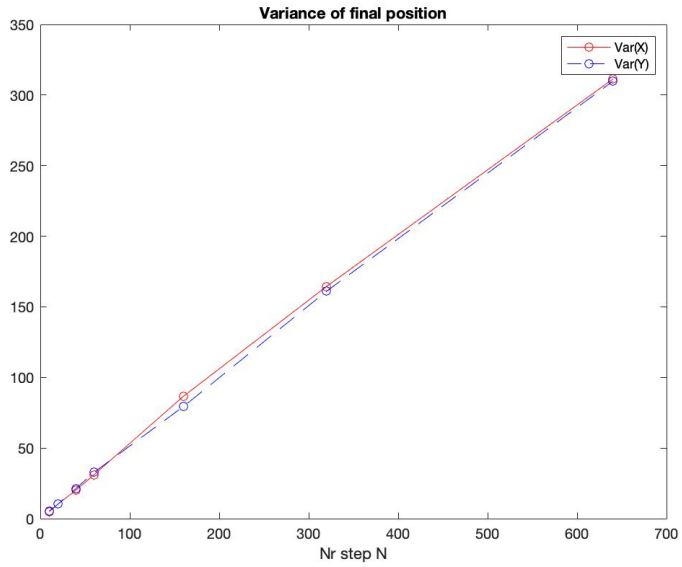

Figure 3: Progression of Var(X) and Var(Y) at fixed number of steps

Figure 3 shows how the square of the average distance along the x and y axis is approximately half of N (number of steps taken), represented by a line of slope $1/2$, which agrees with our theoretical prevision. In particular, by running the code we obtain the following results:

- for N=10, $Var(X) \approx 4.85$, $Var(Y) \approx 4.89$, $Average\ Distance \approx 3.12$
- for N=20, $Var(X) \approx 10.54$, $Var(Y) \approx 9.75$, $Average\ Distance \approx 4.51$
- for N=40, $Var(X) \approx 18.99$, $Var(Y) \approx 21.00$, $Average\ Distance \approx 6.32$
- for N=80, $Var(X) \approx 30.36$, $Var(Y) \approx 32.14$, $Average\ Distance \approx 7.91$
- for N=160, $Var(X) \approx 79.50$, $Var(Y) \approx 77.00$, $Average\ Distance \approx 8.78$
- for N=320, $Var(X) \approx 157.93$, $Var(Y) \approx 1655.60$, $Average\ Distance \approx 12.87$
- for N=640, $Var(X) \approx 333.46$, $Var(Y) \approx 324.49$, $Average\ Distance \approx 18.01$

Conclusions

In conclusion, the experimental results confirm the theoretical prediction of the average distance traveled along each coordinate and the total average distance traveled in 2-D. In addition, plotting several amounts of *random walks* results in a dense figure centered in the origin of the axis, which confirms the symmetry of the probability scheme used to model the movements on the lattice $\mathbb{Z}^2$.

Exercise 2

Introduction

The goal of this exercise is to discretize the differential equation with variable coefficients

$$-(a(x)u'(x))' = f(x), \qquad x \in [0,1]$$

with $0 < a(x) \leq \bar{a}$, subject to homogeneous Dirichlet boundary conditions and Neumann boundary conditions.
The first step is to discretize the domain using a mesh with step size h to identify $n+2$ nodes, and then introduce nodes $x_{j\pm1/2} = x_j \pm \frac{1}{2}h$ for $j = 1, ..., n$. The second step is to use these nodes to define a discretization scheme and write down the resulting linear system. The properties of the coefficient matrix are then investigated, and the discrete maximum principle is derived if possible.

For the case of Neumann boundary conditions, the compatibility condition is written down, and its implications for the solvability of the problem are discussed. The discretization scheme and one-sided derivatives are then used to find the discrete equivalent of the compatibility condition, and its second-order approximation is verified for the case where $a(x)$ is constant.
Finally, the accuracy of the approximations is tested using MATLAB, and the results are discussed.

Analysis

Discretization with Dirichlet boundary conditions The discretization of the differential equation is given by:

$$\frac{1}{h^2}[a_{j-1/2}u_{j-1} + (a_{j-1/2} + a_{j+1/2})u_j - a_{j+1/2}u_{j+1}] = f_j, \quad j = 1, ..., n$$

where h is the step size and $a_{j\pm1/2} = a(x_{j\pm1/2})$. According to homogeneous Dirichlet boundary conditions, we also want $u_0 = u_{n+1} = 0$. This discretization is consistent with the second-order central difference approximation for the second derivative. The corresponding linear system is $\mathbf{Au} = \mathbf{f}$, where $\mathbf{A}$ is the matrix with elements:

$$\begin{bmatrix} a_{1-1/2} + a_{1+1/2} & -a_{1+1/2} & 0 & \cdots & 0 \\ -a_{2-1/2} & a_{2-1/2} + a_{2+1/2} & -a_{2+1/2} & 0 & \cdots \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \cdots & -a_{(n-1)-1/2} & a_{(n-1)-1/2} + a_{(n-1)+1/2} & -a_{(n-1)+1)} \\ 0 & \cdots & 0 & -a_{n-1/2} & a_{n-1/2} + a_{n+1/2} \end{bmatrix}$$

and $\mathbf{f}$ is a vector with elements f_j.
The matrix $\mathbf{A}$ is symmetric and strict diagonally dominant. In addition, we have $a_{jj} > 0$ for each $j = 1, ..., n$, since $\min_{x\in[0,1]} a(x) > 0$; therefore $\mathbf{A}$ is positive definite and invertible which ensures the existence of a unique solution to the linear system of equations. Furthermore, the matrix $\mathbf{A}$ is tridiagonal which makes it easy to solve the system efficiently using the \ MATLAB operator. Hence, we obtain:

$$\mathbf{A} \cdot \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} = \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}$$

Dirichlet code summary Let's get into detail with an example, using $a(x) = e^{-x}$ and $f(x) = 1$, and computing results with MATLAB.
We first use the MATLAB symbolic toolbox and the `dsolve` command to find the exact solution to our Dirichlet problem.

```
syms u(x) a(x) f(x)
Du=diff(u,1);
f=1;
a=exp(-x);
uex=dsolve(-diff((a*Du),1)==f,u(0)==0,u(1)==0);
uex=simplify(uex)
```

We immediately find our exact solution to be $u(x) = e^x(\frac{1}{e-1} - x + 1) - \frac{e}{e-1}$.
We now want to use our discretization model discussed above to find the approximate solution to our problem and compare it with the real one.
Let's follow the MATLAB script `proj2dir.m`, starting with 12 nodes (10 internal nodes + 2 boundaries nodes, `n=10`) and mesh step `h=1/11`. In rows `12-22`, we have some initial definitions and initialization. We also specify Dirichlet boundary conditions. We proceed to create the coefficient matrix using the `spdiags()` function. We define the diagonal and off-diagonal values of the matrix according to the discretization scheme given above. We modify the known term vector `b` to account for the boundary conditions by adding the values at the first and last nodes, which correspond to the boundary values, to the appropriate entries of the vector. Finally, we solve the linear system using the backslash operator and obtain the numerical solution `Uh`, which we plot along with the exact solution `uex`.

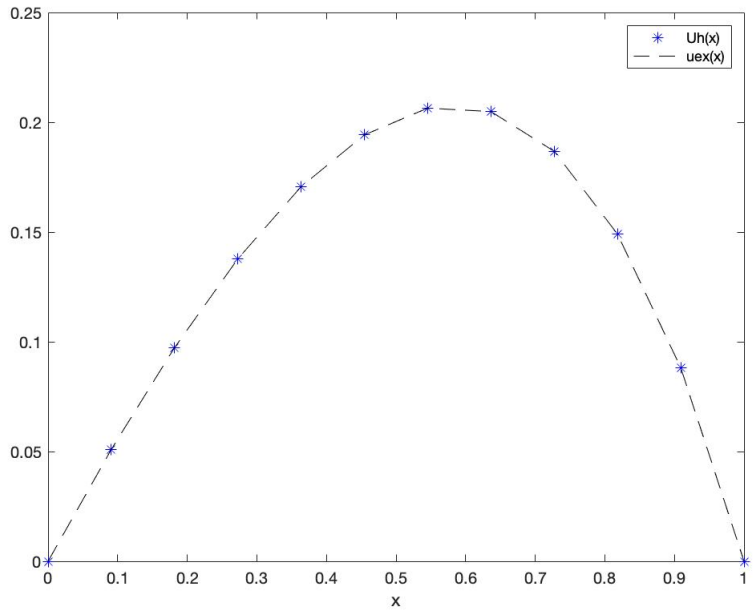

Figure 4: Comparison between exact and approximated solution, with 12 nodes.

As we can see, even considering a small number of nodes, the value of the approximation on each node lies close to the curve of the exact solution. To verify the accuracy of the numerical solution, we can calculate the error norm between the numerical and exact solutions and output the value of h and the error norm.
In the example above, we find `h = 9.0909090909e-02 err = 7.1067865137e-05`, which can be considered accurate enough. We repeat the same procedure for different numbers of nodes to verify the order of accuracy of the numerical scheme. The results are shown in the following table:

Number of spacings (`n`)	Mesh step (`h`)	Error norm (`err`)
10	9.0909e-02	7.1068e-05
20	4.7619e-02	1.9603e-05
50	1.9608e-02	3.3250e-06
100	9.9010e-03	8.4792e-07
200	4.9751e-03	2.1410e-07
500	1.9960e-03	3.4461e-08

Finally, we can verify the maximum principle using the following results:

Theorem. *(Discrete Maximum Principle)*
Let's consider the following ODE: $-\frac{d}{dx}(p(x)\frac{du}{dx}+q(x)u=f(x)$ *for* $x\in(a,b)$.
Let L_h *be our discretization method.*
(i) If $L_hu_j\leq 0$ *for all* $1\leq j\leq n$ *and* $\max(u_0,u_{n+1})\geq 0$, *then* $\max_{0\leq j\leq n+1}u_j\leq\max(u_0,u_{n+1})$.
(ii) If $L_hu_j\geq 0$ *for all* $1\leq j\leq n$, *and* $\min(u_0,u_{n+1})\leq 0$, *then* $\min_{0\leq j\leq n+1}u_j\geq\min(u_0,u_{n+1})$.

Corollary. *If* $q(x)\equiv 0$, *then the discrete maximum principle holds without the hypotheses* $\max(u_0,u_{n+1})\geq 0$ *for part (i) and* $\min(u_0,u_{n+1})\leq 0$ *for part (ii).*

Given that the value of L_hu_j is exactly $f_j=f(x_j)$ for each $j=1,...,n$, where the function $f(x)=e^{-x}$, therefore always positive, the hypothesis are satisfied.

Discretization with Neumann boundary conditions We now want to solve our differential problem, considering the following Neumann boundary conditions:

$$u'(0)=g_0 \qquad u'(1)=g_1$$

with g_0 and g_1 assigned real numbers. We use the same discretization scheme seen above:

$$\frac{1}{h^2}[a_{j-1/2}u_{j-1}+(a_{j-1/2}+a_{j+1/2})u_j-a_{j+1/2}u_{j+1}]=f_j,\quad j=1,...,n$$

However, u_0 and u_{n+1} can't be determined from the boundary conditions, therefore they will be added to the unknown values, components of the vector $\mathbf{u}$.
To incorporate the boundary conditions into the discretization scheme, we use the one-sided finite difference approximations for $u'(0)$ and $u'(1)$, given as:

$$u'(0)\simeq\frac{1}{h}\left(-\frac{3}{2}u(0)+2u(h)-\frac{1}{2}u(2h)\right)$$

$$u'(1)\simeq\frac{1}{h}\left(\frac{1}{2}u(1-2h)-2u(1-h)+\frac{3}{2}u(1)\right)$$

The corresponding linear system is $\mathbf{Au}=\mathbf{f}$, where $\mathbf{A}$ is the matrix with elements:

$$\begin{bmatrix} -3/2 & 2 & -1/2 & 0 & \cdots & 0 \\ -a_{1-1/2} & a_{1-1/2}+a_{1+1/2} & -a_{1+1/2} & 0 & \cdots & 0 \\ 0 & -a_{2-1/2} & a_{2-1/2}+a_{2+1/2} & -a_{2+1/2} & 0 & \cdots \\ 0 & 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \cdots & 0 & -a_{n-1/2} & a_{n-1/2}+a_{n+1/2} & -a_{n+1/2} \\ 0 & \cdots & \cdots & 1/2 & -2 & 3/2 \end{bmatrix}$$

and $\mathbf{f}$ is a vector with elements f_j.
Hence, we obtain:

$$\mathbf{A} \cdot \begin{bmatrix} u_0 \\ \vdots \\ \vdots \\ u_{n+1} \end{bmatrix} = \begin{bmatrix} g_0/h \\ f_1 \\ \vdots \\ f_n \\ g_1/h \end{bmatrix}$$

The values $g0/h$ and $g1/h$ in the known terms vector result from the incorporation of discretized Neumann boundary conditions into our system of equations. To verify the accuracy order of the one-sided derivative formula for $u'(0)$, we need to perform a Taylor series expansion of $u(x)$ around $x = 0$. We have:

$$u(h) = u(0) + hu'(0) + \frac{h^2}{2}u''(0) + o(h^3)$$

$$u(2h) = u(0) + 2hu'(0) + 2h^2u''(0) + o(h^3)$$

Substituting these expressions into the one-sided derivative formula, we get:

$$\begin{aligned} u'(0) &\simeq \frac{1}{h}\left(-\frac{3}{2}u(0) + 2u(h) - \frac{1}{2}u(2h)\right) \\ &= \frac{1}{h}\left(-\frac{3}{2}u(0) + 2u(0) + 2hu'(0) + h^2u''(0) - \frac{1}{2}u(0) - hu'(0) - h^2u''(0) + o(h^3)\right) \\ &= u'(0) + o(h^2) \end{aligned}$$

Therefore, the error of the formula for $u'(0)$ is given by:

$$u'(0) - \frac{1}{h}\left(-\frac{3}{2}u(0) + 2u(h) - \frac{1}{2}u(2h)\right) = o(h^2)$$

which implies that the one-sided derivative formula for $u'(0)$ has an accuracy of $o(h^2)$.
Similarly, for $u'(1)$ we have:

$$u(1-h) = u(1) - hu'(1) + \frac{h^2}{2}u''(1) + o(h^3)$$

$$u(1-2h) = u(1) - 2hu'(1) + 2h^2u''(1) + o(h^3)$$

Substituting these expressions into the one-sided derivative formula, we get:

$$\begin{aligned} u'(1) &\simeq \frac{1}{h}\left(\frac{3}{2}u(1) - 2u(1-h) + \frac{1}{2}u(1-2h)\right) \\ &= \frac{1}{h}\left(\frac{3}{2}u(1) - 2u(1) + 2hu'(1) - h^2u''(1) + \frac{1}{2}u(1) - hu'(1) + h^2u''(1) + o(h^3)\right) \\ &= u'(1) + o(h^2) \end{aligned}$$

Therefore, the error of the formula for $u'(1)$ is given by:

$$u'(1) - \frac{1}{h}\left(\frac{3}{2}u(1) - 2u(1-h) + \frac{1}{2}u(1-2h)\right) = o(h^2)$$

which implies that the one-sided derivative formula for $u'(1)$ has again an accuracy of $o(h^2)$.
To guarantee the existence of a unique solution, we have to satisfy the *compatibility condition*.

The *compatibility condition* for the differential equation $-(a(x)u'(x))' = f(x)$ can be obtained by integrating the differential equation over the domain $[0,1]$ and using integration by parts:

$$\begin{aligned}
\int_0^1 -\left(a(x)u'(x)\right)' dx &= \int_0^1 f(x)dx \\
-\left[a(x)u'(x)\right]_0^1 &= \int_0^1 f(x)dx \\
a(0)u'(0) - a(1)u'(1) &= \int_0^1 f(x)dx \\
g_0 a(0) - g_1 a(1) &= \int_0^1 f(x)dx \qquad (*)
\end{aligned}$$

where we have used the boundary conditions $u'(0) = g_0$ and $u'(1) = g_1$ in the third line.
Considering $n+2$ nodes and $a(x) = \bar{a} \in \mathbb{R}$, we obtain the following discrete version of $(*)$:

$$\bar{a}(g_0 - g_1) = \sum_{j=0}^{n} f\left(x_j\right) \frac{1}{n+1} = h \sum_{j=0}^{n} f\left(x_j\right) \quad (**)$$

where g_0 and g_1 are the one-sided derivatives discussed above. Since g_0 and g_1 are second-order approximations and $\bar{a}$ is a constant, the $(**)$ is also a second-order approximation of $(*)$.

Neumann code summary Let's get into detail using again $a(x) = e^{-x}$ and $f(x) = 1$, and computing results with MATLAB.
As we did for Dirichlet BCs, we use the MATLAB symbolic toolbox and the `dsolve` command to find the exact solution to our Neumann problem. The algorithm can be found on the `exactsol.m` MATLAB script.

```
syms u(x) a(x) f(x)
Du=diff(u,1);
f=1;
a=exp(-x);
temp_uex=dsolve(-diff((a*Du),1)==f);
temp_uex=simplify(temp_uex);
% uex=vectorize(uex)
temp_uex
temp_duex=diff(temp_uex,1)
der_value_in0=subs(temp_duex,x,0)
C1=0 % value of integration constant. Can be seen as u'(0).
uex = exp(x)*(C1 - x + 1);
duex=diff(uex,1)
der_value_in1=subs(duex,x,1)
```

We arbitrarily choose the value of g_0 (`C1` in the script) to be 0. Our computation shows how the value of g_1 is then forced to be $-e$. Using our analytic results, from $(*)$ we have:

$$g_0 e^0 - g_1 e^{-1} = \int_0^1 1dx$$

hence, $g_1 = (g_0 - 1)e = (0-1)e = -e$, as found using MATLAB.
From now on, we will assume $(g_1, g_2) = (0, -e)$.
We find our exact solution to be $u(x) = e^x(1-x)$. We now want to use the discretization discussed above to find the approximate solution to our Neumann problem and compare it with the real one.

Let's follow the MATLAB script `proj2neum.m`, starting with 12 nodes (10 internal nodes + 2 boundaries nodes, `n=10`) and mesh step `h=1/11`. In rows `19-29`, we have some initial definitions and initialization. We proceed to create the coefficient matrix using the `spdiags()` function. We define the diagonal and off-diagonal values of the matrix according to the discretization scheme given above. This time, we will have two extra rows that incorporate the equation to represent the one-sided approximation of $u'(0) = g_0 = 0$ and $u'(1) = g_1 = -e$. We modify the known term vector `b` to account for the boundary conditions by adding the values g_0/h and g_1/h to the appropriate entries of the vector. Finally, we solve the linear system using the backslash operator and obtain the numerical solution `Uh`, which we plot along with the exact solution `uex`.

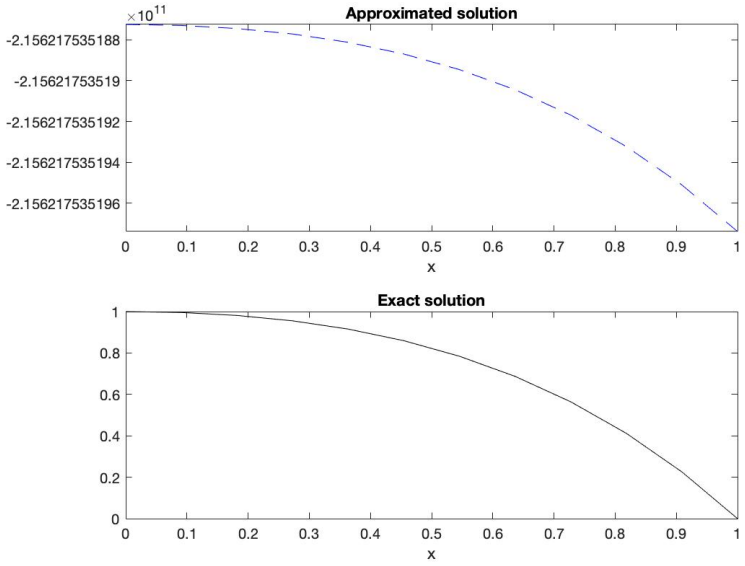

Figure 5: Comparison between exact and approximated solution, with 12 nodes.

We can easily notice how our solutions both respect the Neumann boundaries condition. However, they differ from a constant. In fact, the generic solution to our problem is $C_2 + e^x(C_1 - x + 1)$. By imposing the Neumann BCs - differentiating the solution - we lose the information on the additive constant C_1. This results in an infinite set of solutions that satisfy our boundary problem, even though they are all translated by a constant.
To have a significantly comparable pair of solutions, we need to add a Dirichlet boundary condition in $x = 0$. Results of this implementation can be found on the MATLAB script `proj2neum_dirichlet.m`. Plotting both the approximated and exact solution, we now have:

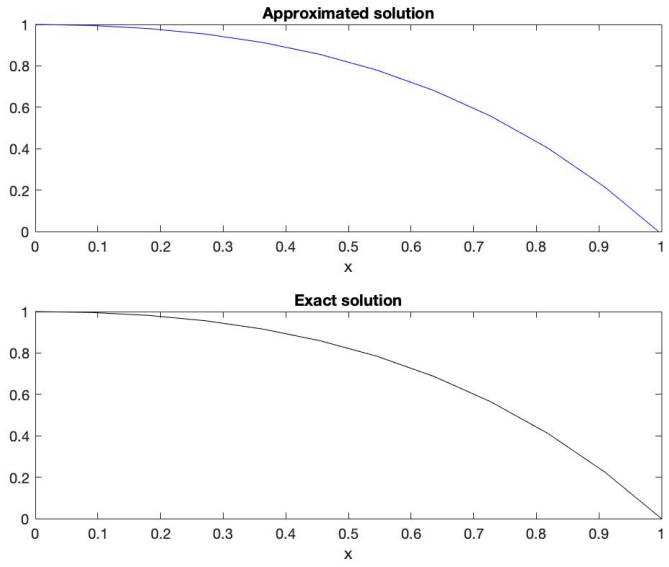

Figure 6: Comparison between exact and approximated solution, with Neumann+Dirichlet BCs.

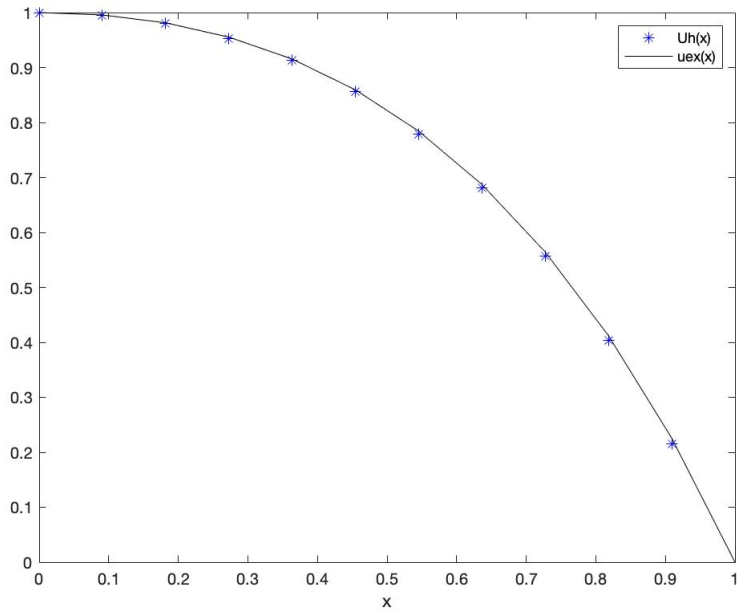

Figure 7: Comparison between exact and approximated solution, with 12 nodes highlighted.

To verify the accuracy of the numerical solution, we can calculate the error norm between the numerical and exact solutions as we did with the Dirichlet problem, and output the value of h and the error norm. In the example above, we find `h = 9.0909090909e-02 err = 1.2066273753e-02`, which can be considered accurate enough. We repeat the same procedure for different numbers of nodes to verify the order of accuracy of the numerical scheme. The results are shown in the following table:

Number of spacings (`n`)	Mesh step (`h`)	Error norm (`err`)
10	9.0909e-02	1.2066e-02
20	4.7619e-02	3.3268e-03
50	1.9608e-02	5.6642e-04
100	9.9010e-03	1.4468e-04
200	4.9751e-03	3.6566e-05
500	1.9960e-03	5.8893e-06

Conclusions

In conclusion, the proposed discretization method appears to be quite accurate. However, the accuracy of the method was found to be much higher in the Dirichlet boundary problem, compared to the Neumann boundary problem. For instance, when considering a node number of $100 + 2$ ($\mathtt{n} = 100$), in the former case, the error is of the order of 10^{-7}, whereas in the latter case, it is of the order of 10^{-4}.