$$-\frac{v_{i+1}+v_{i-1}-2v_i}{h^2}=f_i \qquad \text{for } i=1,\dots,n, \tag{1}$$

where $f_i = f(x_i)$.

Rewriting the Equation 1 as a linear set of equations of the form

$$\mathbf{A}\mathbf{v} = \tilde{\mathbf{b}},$$

where $\mathbf{A}$ is an $n \times n$ tridiagonal matrix which we rewrite as

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 & 2 & -1 \\ 0 & 0 & \cdots & 0 & -1 & 2 \end{bmatrix},$$

and $\tilde{b}_i = h^2 f_i$.

Writing out the equations:

$$\begin{aligned} i=1 &: -v_0 + 2v_1 - v_2 = f_1 h^2 \\ i=2 &: -v_1 + 2v_2 - v_3 = f_2 h^2 \\ &\vdots \\ i=n &: -v_{n-1} + 2v_n - v_{n+1} = f_n h^2 \end{aligned}$$

where $v_0 = 0$ and $v_{n+1} = 0$.

Adding some zeros to the equations for illustrative purposes:

$$\begin{aligned} &i=1: && 2v_1 - v_2 + 0 + 0 + 0 + 0 + \cdots + 0 = f_1 h^2 \\ &i=2: && -v_1 + 2v_2 - v_3 + 0 + 0 + \cdots + 0 = f_2 h^2 \\ &i=3: && 0 - v_2 + 2v_3 - v_4 + 0 + \cdots + 0 = f_3 h^2 \\ &\vdots && \vdots \\ &i=n-1: && 0 + \cdots + 0 - v_{n-2} + 2v_{n-1} - v_n = f_{n-1} h^2 \\ &i=n: && 0 + \cdots + 0 + 0 + 0 - v_{n-1} + 2v_n = f_n h^2 \end{aligned}$$

We can now recognize the equations as a product between a matrix an a vector that equals another vector.

Defining vectors that fit:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \qquad \tilde{\mathbf{b}} = \begin{bmatrix} f_1 h^2 \\ f_2 h^2 \\ \vdots \\ f_n h^2 \end{bmatrix}$$

The matrix is the matrix $\mathbf{A}$. Then we have:

$$\begin{bmatrix} 2 & -1 & 0 & \cdots & 0 & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & -1 & 2 & -1 \\ 0 & 0 & \cdots & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{n-1} \\ v_n \end{bmatrix} = \begin{bmatrix} f_1 h^2 \\ f_2 h^2 \\ f_3 h^2 \\ \vdots \\ f_{n-1} h^2 \\ f_n h^2 \end{bmatrix} \implies \mathbf{A}\mathbf{v} = \tilde{\mathbf{b}}$$

Putting the solution into the Poisson equation:

The Poisson equation:

$$-u''(x) = f(x)$$

In this case $f(x) = 100e^{-10x}$.

The solution and derivatives:

$$\begin{aligned} u(x) &= 1 - \left(1 - e^{-10}\right)x - e^{-10x} \\ u'(x) &= \left(1 - e^{-10}\right) + 10e^{-10x} \\ u''(x) &= -100e^{-10x} \end{aligned}$$

The solution is a solution of the Poisson equation:

$$-u''(x) = -(-100)e^{-10x} = 100e^{-10x} = f(x)$$

A list of what we are supposed to do from the exercise text:

- Set up the general algorithm (assuming different values for the matrix elements) for solving this set of linear equations.
- Find also the precise number of floating point operations needed to solve the above equations.
- Code the above algorithm and solve the problem for matrices of the size $10 \times 10$, $100 \times 100$ and $1000 \times 1000$. That means that you select $n = 10$, $n = 100$ and $n = 1000$ grid points.
- Compare your results (make plots) with the closed-form solution for the different number of grid points in the interval $x \in (0, 1)$. The different number of grid points corresponds to different step lengths $h$.

A list of what we are supposed to do from the exercise text:

- Use thereafter the fact that the matrix has identical matrix elements along the diagonal and identical (but different) values for the non-diagonal elements. Specialize your algorithm to the special case and find the number of floating point operations for this specific tri-diagonal matrix.
- Compare the CPU time with the general algorithm from the previous point for matrices up to $n = 10^6$ grid points.

A list of what we are supposed to do from the exercise text:

- Compute the relative error in the data set $i = 1, \ldots, n$,by setting up

  $$\epsilon_i = log_{10}\left(\left|\frac{v_i - u_i}{u_i}\right|\right),$$

  as function of $log_{10}(h)$ for the function values $u_i$ and $v_i$. For each step length extract the max value of the relative error.
- Increase $n$ to $n = 10^7$ and make a table of the results and comment your results. You can use either the algorithm from b) or c).

A list of what we are supposed to do from the exercise text:

- Compare your results with those from the LU decomposition codes for the matrix of sizes $10 \times 10$, $100 \times 100$ and $1000 \times 1000$ (use the library functions provided on the webpage of the course. Alternatively, if you use armadillo as a library, you can use the similar function for LU decomposition. The armadillo function for the LU decomposition is called *LU* while the function for solving linear sets of equations is called *solve*. Use for example the unix function *time* when you run your codes and compare the time usage between LU decomposition and your tridiagonal solver. Alternatively, you can use the functions in C++, Fortran or Python that measure the time used)
- Make a table of the results and comment the differencesin execution time.
- How many floating point operations does the LU decomposition use to solve the set of linear equations?
- Answer: Can you run the standard LU decomposition for a matrix of the size $10^5 \times 10^5$?
- Comment your results.

  To compute the elapsed time in c++ you can use the following statements (see comment)