title goes here

introduction

a)

We start with the given equation

$-\frac{v_{i-1}-2v_i+v_{i+1}}{h^2} = f_i$

where $i \in [1,n] \cap \mathbb{N}$ and $n \in \mathbb{N}$. We will assume that $n \geqslant 3$ and that $v_0 = v_{n+1} = 0$.

$-v_{i-1} + 2v_i - v_{i+1} = h^2 f_i \equiv \widetilde{b}_i$

$$\begin{bmatrix} -1 & 2 & -1 \end{bmatrix} \begin{bmatrix} v_{i-1} \\ v_i \\ v_{i+1} \end{bmatrix} = \widetilde{b}_i$$

Now we expand these vectors from 3 elements to n elements. Note that the ith element of the row vector should be 2 and the ith element of the column vector should be v_i after the expansion:

$$\begin{bmatrix} 0 & \cdots & -1 & 2 & -1 & \cdots & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ \vdots \\ v_{i-1} \\ v_i \\ v_{i+1} \\ \vdots \\ v_n \end{bmatrix} = \widetilde{b}_i$$

Note that in the row vector 2 can very well be the first or last element, in which case the preceding presentation can be a little misleading. We further recognize the row vector as the ith row of the $n \times n$ matrix $\mathbf{A}$ (defined such that $A_{ij} = 2\delta_{ij} - \delta_{i(j-1)} - \delta_{i(j+1)}$) and get the inner product

$\mathbf{A}_i \cdot \mathbf{v} = \widetilde{b}_i$

and remembering that $i \in [1,n]$, by the definition of matrix multiplication

$\mathbf{Av} = \widetilde{\mathbf{b}}$

which is what we wanted to show.

b)

The algorithm we will use is as follows: First we do forward subsitution by subtracting from the current row a multiple of the row before it.

$A_i = A_i - x_i A_{i-1}$

Here, x_i is the factor that cause the term a_i to cancel out. Before the first step, the rows $i-1$ and i look like this (the * indicates a value that has been changed by the algoritm)

$$\begin{array}{ccccccc} \cdots & 0 & b^*_{i-1} & c_{i-1} & 0 & 0 & \cdots \\ \cdots & 0 & a_i & b_i & c_i & 0 & \cdots \end{array}$$

and our goal is that it look like this after the forward substitution

$$\begin{array}{ccccccc} \cdots & 0 & b^*_{i-1} & c_{i-1} & 0 & 0 & \cdots \\ \cdots & 0 & 0 & b^*_i & c_i & 0 & \cdots \end{array}$$

(note that c_i is unchanged). When we reach the bottom, we do a backward subsitution by adding to the current row a multiple of the row below it and then dividing the last element by itself to make the last element equal to 1:

$A_i = A_i - c_i A_{i+1}$

$A_i = \frac{A_i}{b^*_i}$

That is to say, we go from

$$\begin{array}{ccccccc} \cdots & 0 & b^*_i & c_i & 0 & \cdots \\ \cdots & 0 & 0 & 1 & 0 & \cdots \end{array}$$

to

$$\begin{array}{ccccccc} \cdots & 0 & 1 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 1 & 0 & \cdots \end{array}$$

Naturally, we also have to do equivalent operations on the vector $\widetilde{\mathbf{b}}$ on the other side of the equation. We will not calculate any unnecessary values, i.e. values

that will not be used by the program later on. Hence the algoritm looks like this (number of flops in parantheses):

For $i : 2 \longrightarrow n$

1. $x_i = \frac{a_i}{b^*_{i-1}}$ (n)
2. $b_i = b_i - x_i c_{i-1}$ $(2n)$
3. $\widetilde{b}_i = \widetilde{b}_i - x_i \widetilde{b}_{i-1}$ $(2n)$

For $i : n-1 \longrightarrow 1$

1. $\widetilde{b}_i = \widetilde{b}_i - c_i \widetilde{b}_{i+1}$ $(2n)$
2. $\widetilde{b}_i = \frac{\widetilde{b}_i}{b_i}$ (n)

This makes the total running time of the algorihm $8n$ (actually, it is $8(n-1)$, but we are mostly concerned with the performance for large n, where $n \approx n-1$). If the matrix is symmetric, so that $a_i = c_{i-1}$, then $x_i c_{i-1} = \frac{a_i^2}{b^*_{i-1}}$. If $a_i = c_{i-1} = k$, a constant value which is the same for every row, where $k \in \{1,\text{-}1\}$, then $k^2 = 1$ and $x_i c_{i-1} = \frac{1}{b^*_{i-1}} = x_i$. Furthermore, $c_i \widetilde{b}_{i+1} = k\widetilde{b}_{i+1} = \pm\widetilde{b}_{i+1}$ (depending on the sign of k). The algorithm will then look like this:

For $i : 2 \longrightarrow n$

1. $x_i = \frac{1}{b^*_{i-1}}$ (n)
2. $b_i = b_i - x_i$ (n)
3. $\widetilde{b}_i = \widetilde{b}_i - x_i \widetilde{b}_{i-1}$ $(2n)$

For $i : n-1 \longrightarrow 1$

1. $\widetilde{b}_i = \widetilde{b}_i \pm b_{i+1}$ (n)
2. $\widetilde{b}_i = \frac{\widetilde{b}_i}{b_i}$ (n)

c)

...