1 Theory

1.1 The one-dimensional Poisson equation

We consider the one-dimensional Poisson equation with Dirichlet boundary conditions. That is, we are given a function $f\colon [0,1] \to \mathbb{R}$ and seek a function $u\colon [0,1] \to \mathbb{R}$ such that

$$-u''(x) = f(x), \text{ with } u(0) = u(1) = 0. \tag{1}$$

To attack this problem, we do the following. We fix a positive integer n and let $h := 1/(n+1)$. Then we partition $[0,1]$ into $n+1$ subintervals $[x_i, x_{i+1}]$ such that $x_i = ih$ for all $i = 0, \dots, n+1$. Observe that the length of each interval is given by $h := 1/(n+1)$. Assuming that u is an exact solution to eq. (1), we will find an approximation to u which we label v. The function v will be a function $v\colon [0,1] \to \mathbb{R}$ such that v is linear on each interval $[x_i, x_{i+1}]$ and such that $v(x_i) \approx u(x_i)$ for all i.

Our task is to find an ordered set of points $\{v_0, \dots, v_{n+1}\}$ such that $v_i \approx u(x_i)$. Then $v(x_i) := v_i$ determines the approximation v completely.

1.2 Approximation to $u''(x)$

We now derive an approximation to $u''(x)$ using Taylor expansions. Consider the taylor expansions of u about x_i:

$$u(x_i \pm h) = u(x_i) \pm u'(x_i)h + \frac{u''(x_i)h^2}{2} + \mathcal{O}(h^3)$$

It is seen from adding $u(x_i + h)$ and $u(x_i - h)$ that:

$$u''(x_i) = \frac{u(x_{i+1}) + u(x_{i-1}) - 2u(x_i)}{h^2} + \mathcal{O}(h^2)$$

Assuming that we have discretized u and consider the points $u_i := u(x_i)$, we can define a discrete axpproximation to $u''(x_i)$ as

$$\frac{u_{i+1} + u_{i-1} - 2u_i}{h^2}$$

for $i = 1, \dots, n$.

We now reformulat eq. (1) in terms of the approximation of $u''(x)$. Let $f_i = f(x_i)$. Then eq. (1) becomes:

$$-\frac{v_{i+1} + v_{i-1} - 2v_i}{h^2} = f_i, \quad v_0 = v_{n+1} = 0, \quad i = 1, \dots, n \tag{2}$$

Solving this set of linear equations yields the ordered set $\{v_0, \dots, v_{n+1}\}$ that we are after. The aim now is to solve this set of linear equations and obtain the approximation v to u.

1.3 The one dimensional Poisson equation as a matrix equation

Consider eq. (2). Multiplying both sides by h^2 yields the equivalent set of linear equations:

$$2v_i - v_{i+1} - v_{i-1} = h^2 f_i, \quad i = 1, \dots, n$$

Because we assume $v_0 = v_{n+1} = 0$, we can discard the equations that would otherwise be needed to solve for v_0 and v_n. In other words, we may discard the equations corresponding to $i = 1$ and $i = n$.

We want to reformulate the remaining equations ($1 < i < n$) into a matrix equation. We define the $n \times n$ matrix

$$A := \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & -1 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix}$$

and the vectors $v := (v_1, \dots, v_n)$ and $d := h^2(f_1, \dots, f_n)$. We have

$$Av = d, \tag{3}$$

as is seen from taking the dot product of the i'th row in A with v.

1.4 An analytical solution to the one dimensional Poisson equation

Assume that $f(x) = 100e^{-10x}$. We now show that $u(x) = 1 - (1 - e^{-10})x - e^{-10x}$ is an analytical solution to eq. (1) by insertion. We have

$$u'(x) = 1 - e^{-10} + 10e^{-10x}$$
$$u''(x) = -100e^{-10x}$$

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

Remark 1.1. The above solution will be a point of reference for the numerical solution, so that we may verify the validity of the proposed numerical solution. That is, we shall compare the approximation v to u for different values of n.

1.5 Solving a general tridiagonal matrix equation

A **tridiagonal matrix** is a square matrix whose only nonzero entries are the entries along the main diagonal and the entries along the diagonal directly above and the diagonal directly below the main diagonal.

Let temporarily A be any $n \times n$ tridiagonal matrix with $n \geq 2$ and v and d be two n-dimensional vectors. Note that

$$A = \begin{pmatrix} b_1 & c_1 & 0 & \cdots & 0 \\ a_1 & b_2 & c_2 & \ddots & \vdots \\ 0 & a_2 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & c_{n-1} \\ 0 & \cdots & 0 & a_{n-1} & b_n \end{pmatrix}$$

is uniquely determined by three vectors $a = (a_1, \dots, a_{n-1})$, $b = (b_1, \dots, b_n)$, and $c = (c_1, \dots, c_{n-1})$.

We seek to solve an equation of the form $Av = d$. That is, we seek to solve:

$$\begin{pmatrix} b_1 & c_1 & 0 & \cdots & 0 \\ a_1 & b_2 & c_2 & \ddots & \vdots \\ 0 & a_2 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & c_{n-1} \\ 0 & \cdots & 0 & a_{n-1} & b_n \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ \vdots \\ v_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ \vdots \\ d_n \end{pmatrix} \tag{4}$$

This can be accomplished by row reducing the matrix:

$$\begin{pmatrix} b_1 & c_1 & 0 & \cdots & 0 & d_1 \\ a_1 & b_2 & c_2 & \ddots & \vdots & \vdots \\ 0 & a_2 & \ddots & \ddots & 0 & d_{n-2} \\ \vdots & \ddots & \ddots & \ddots & c_{n-1} & d_{n-1} \\ 0 & \cdots & 0 & a_{n-1} & b_n & d_n \end{pmatrix} \sim \begin{pmatrix} \beta_1 & c_1 & 0 & \cdots & 0 & \delta_1 \\ 0 & \beta_2 & c_2 & \ddots & \vdots & \vdots \\ \vdots & 0 & \ddots & \ddots & 0 & \delta_{n-2} \\ \vdots & \ddots & \ddots & \ddots & c_{n-1} & \delta_{n-1} \\ 0 & \cdots & \cdots & 0 & \beta_n & \delta_n \end{pmatrix}$$

Now we need to rid of the a_i's one by one. The first row operation needed is then clearly $R_2 \to R_2 - \frac{a_1}{b_1} R_1$. And then we wish to generalize this pattern further. The general algorithm for this row reduction can be described as follows. We inductively update each row accordingly, and one at the time starting from above:

Algorithm 1. (Row reduction)

(i) The initial steps are $\beta_1 := b_1$ and $\delta_1 := d_1$.

(ii) Let $\epsilon_i := a_{i-1}/\beta_{i-1}$. The inductive step is:

$$\beta_i := b_i - \epsilon_i c_{i-1}, \quad \text{and } \delta_i := d_i - \epsilon_i \delta_{i-1}, \quad \text{for } i = 2, \ldots, n$$

Remark 1.2. The number of floating point operations needed to run the above algorithm is $7(n-1)$. Indeed, (i) requires no floating point operations and there are $n-1$ steps left. In each step we need to do 2 times a calculation of the form $a - b * c$ and one times a calculation of the form a/b, yielding a total of 7.

Equation (4) is now equivalent to the following set of linear equations in the variables $v_0, \ldots, v_{n+1}$:

$$\beta_{i-1} v_{i-1} + c_{i-1} v_i = \delta_i, \quad \text{for } i = 1, \ldots, n-1$$
$$\beta_n v_n = \delta_n$$

The algorithm to solve these equations becomes:

Algorithm 2. (Solve equations)

(i) The initial step is $v_n = \delta_n / \beta_n$.

(ii) The inductive step is:

$$v_i = \frac{\delta_{i+1} - c_i v_{i+1}}{\beta_i}, \quad \text{for } i = n-1, ..., 1$$

Remark 1.3. By a counting argument similar to the one in remark 1.2, the number of floating point operations needed for the above algorithm is $1 + 3(n-1)$.

1.6 The tridiagonal matrix equation for $-u''(x) = f(x)$

The special case we seek to solve is:

$$A := \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & -1 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ \vdots \\ v_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ \vdots \\ d_n \end{pmatrix}$$

We can accomplish this by simplifying the algorithms described in section 1.5. Note that $\epsilon_i = 1/2$ for all $i = 2, \ldots, n$. Then row reducing A immediately yields

$$\begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ 0 & 3 & -1 & \ddots & \vdots \\ 0 & 0 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & 0 & 0 & 3 \end{pmatrix},$$

and we immediately know β_i for all $i = 1, \ldots, n$. Hence the number of floating point operations needed to perform this algorithm is reduced significantly from the general case. This is elaborated below.

From algorithm 1 we only need part (ii), which reads:

$$\delta_i = d_i - \frac{1}{2}\delta_{i-1}, \quad \text{for } i = 2, \ldots, n$$

The number of flops then becomes $2(n-1)$.

From algorithm 2 we get:

(i) $v_n = \delta_n/3$.

(ii) $v_i = (\delta_{i+1} + v_{i+1})/3, \quad \text{for } i = n-1, ..., 1$

which corresponds to $1 + 2(n-1)$ flops.