Contents

The algorithm

In order to solve the one-dimensional Poisson equation

$$-u''(x) = f(x) \tag{1}$$

with Dirichlet boundary conditions in the interval $(0,1)$ we rewrite the latter as a set of linear equations by discretizing the problem. In this way we obtain a set of n grid points with the gridwidth $h = 1/(n+1)$ .Then we approximate the second derivative u"(x) with

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

From this we can easily derive the following matrix equation:

$$\begin{pmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots \\ 0 & -1 & 2 & -1 & 0 & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & & -1 & 2 & -1 \\ 0 & \dots & & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \dots \\ \dots \\ \dots \\ v_n \end{pmatrix} = \begin{pmatrix} h^2 f_1 \\ h^2 f_2 \\ \dots \\ \dots \\ \dots \\ h^2 f_n \end{pmatrix}. \tag{3}$$

A more general form of the above is the following:

$$\begin{pmatrix} a_0 & b_0 & 0 & \dots & \dots & \dots \\ c_1 & a_1 & b_1 & \dots & \dots & \dots \\ & c_3 & a_2 & b_2 & \dots & \dots \\ & \dots & \dots & \dots & \dots & \dots \\ & & & a_{n-2} & b_{n-1} & c_{n-1} \\ & & & & a_n & b_n \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \dots \\ \dots \\ \dots \\ v_n \end{pmatrix} = \begin{pmatrix} w_1 \\ w_2 \\ \dots \\ \dots \\ \dots \\ w_n \end{pmatrix}. \tag{4}$$

Since the Gaussian elimination would of course lead to the rights results here, the execution time can be easily reduced from $\sim n^3$ to $\sim n$ by applying an algorithm that no longer requires the matrix but uses the three diagonals as arrays. In other words we consider that the rest of the matrix is 0 everywhere except for these diagonals which the brute force Gaussian elimination way does not take in account. The following steps have then to be taken:

1. The three diagonals are stored in arrays $a[]$, $b[]$, and $c[]$, as well as the right side of the equation is stored in an array $w[]$ of the size n. $c[0]$ and $b[n-1]$ are set to 0.

2. Then the entries in $a[]$ are substituted recursively by

$$\tilde{a}[0] = a[0], \quad \tilde{a}[i] = a[i] - b[i-1]\frac{c[i]}{\tilde{a}[i-1]} \tag{5}$$

   This requires $3 \cdot (n-1)$ floating point operations for we obtain a division, a substraction and a multiplication for each substitution.

3. Accordingly $w[]$ is substituted by

$$\tilde{w}[0] = a[0], \quad \tilde{w}[i] = w[i] - \tilde{w}[i-1]\frac{c[i]}{\tilde{a}[i-1]} \tag{6}$$

   This only requires $2 \cdot (n-1)$ flops for we already did the division $\frac{c[i]}{\tilde{a}[i-1]}$ during the substitution above.

4. Finally backward substitution is used to gain the result for the unknown vector v which is stored in another array $v[]$:

$$v[n-1] = \frac{\tilde{w}[n-1]}{\tilde{a}[n-1]}, \quad v[i] = \frac{\tilde{w}[i] - b[i] \cdot v[i+1]}{\tilde{a}[i-1]} \tag{7}$$

   This operation results in another $3 \cdot (n-1) + 1$ flops for we have again a substraction, a multiplication and a division for each resubstitution plus a division for the first element.

In sum the algorithm needs $8 \cdot (n-1) + 1$ floating point operations to solve the general matrix equation 4.