

Gaussian elimination for tridiagonal linear systems

The general linear system of n equations can be written in a matrix form as follows

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ \vdots \\ b_n \end{bmatrix}. \quad (1)$$

But many applications lead to the linear systems which have the so called *banded structure*. This means that nonzero elements in the left hand side of (1) appear only around the main diagonal. Such condition may be expressed as follows

$$\text{there exist } m \ll n, \text{ such that } a_{ij} = 0 \text{ for } |i - j| > m.$$

The above condition means that nonzero elements reside only on the band of the $2m+1$ width around the main diagonal. For such systems it is not efficient to employ full version of the Gaussian eliminations as most operations would simply be carried out on the zero elements.

Now we will concentrate on the banded system which have only three diagonals which may be occupied by nonzero entries. Such system arises in the numerical methods for solving boundary problem for one dimensional Poisson's equation

$$-u'' = f(x), \quad (2)$$

or more generally second order linear equation with variable coefficients

$$p(x)u'' + q(x)u' + r(x)u = f(x), \quad (3)$$

for $x \in I \subset \mathbb{R}$, where I denotes some interval.

The tridiagonal system can be written in a matrix form as

$$\begin{bmatrix} d_1 & c_1 & 0 & 0 & \dots & 0 \\ a_1 & d_2 & c_2 & 0 & \dots & 0 \\ 0 & a_2 & d_3 & c_3 & \ddots & \vdots \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & a_{n-2} & d_{n-1} & c_{n-1} \\ 0 & 0 & \dots & 0 & a_{n-1} & d_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix}. \quad (4)$$

For the description of Gaussian elimination we do not need the column of unknowns, so for the sake of simplicity we use the following description of (4)

$$\left[\begin{array}{cccccc|c} d_1 & c_1 & 0 & 0 & \dots & 0 & b_1 \\ a_1 & d_2 & c_2 & 0 & \dots & 0 & b_2 \\ 0 & a_3 & d_3 & c_3 & \ddots & \vdots & b_3 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 & \vdots \\ \vdots & \vdots & \ddots & a_{n-2} & d_{n-1} & c_{n-1} & b_{n-1} \\ 0 & 0 & \dots & 0 & a_{n-1} & d_n & b_n \end{array} \right]. \quad (5)$$

The elimination is now a process which converts this extended matrix into upper triangular one, what means that below the main diagonal are only zeroes. Of course, this process is restricted to such operations that leave the solution of the system (4) unchanged. There are two basic operations that satisfy this requirement and are enough to bring our system to the triangular form: (i) adding any row multiplied by a number to another row; (ii) swapping two rows.

Gaussian elimination method for the tridiagonal system (5) has the following basic structure

$$\left\{ \begin{array}{l} R_1 - \text{unchanged}, \\ R_2 := R_2 - \frac{a_1}{d_1} R_1, \\ R_3 := R_3 - \frac{a_2}{d_2} R_2, \\ \vdots \\ R_n := R_n - \frac{a_{n-1}}{d_{n-1}} R_{n-1}, \end{array} \right. \quad (6)$$

where R_1, \dots, R_n are the rows of the extended matrix (5). But we should also take into account the fact that rows R_1, \dots, R_n contain only two or three nonzero entries so the subtraction of rows in (6) basically is reduced to two values:

$$\left\{ \begin{array}{l} d_1, c_1 - \text{unchanged} \\ d_2 = d_2 - \frac{a_1}{d_1} c_1, b_2 = b_2 - \frac{a_1}{d_1} b_1, \\ d_3 = d_3 - \frac{a_2}{d_2} c_2, b_3 = b_3 - \frac{a_2}{d_2} b_2, \\ \vdots \\ d_n = d_n - \frac{a_{n-1}}{d_{n-1}} c_{n-1}, b_n = b_n - \frac{a_{n-1}}{d_{n-1}} b_{n-1}. \end{array} \right.$$

Using the pseudo C/C++ code this fragment reads as (remember, however, that tables in C/C++ have indices that start naturally rather *from zero* than from one thus n – elements tables are indexed as d_0, d_1, \dots, d_{n-1})

```
double a[n], d[n], c[n],
```

```

for (i=1; i < n; i++) {
    d[i] = d[i] - (a[i-1]/d[i-1])*c[i-1];
    b[i] = b[i] - (a[i-1]/d[i-1])*b[i-1];
}

```

Once we have arrived at the upper triangle form we can now easily compute the solutions by the simple backwards substitution. But for the case at hand our upper triangle system is not a full one because in each row there are only two nonzero elements d_i, c_i (except for the last row where there is only d_n) so we can use the restricted form of the general procedure which now will require only one operation per one unknown variable. Specifically, the system looks like below

$$\begin{cases} d_1 x_1 + c_1 x_2 = b_1, \\ d_2 x_2 + c_2 x_3 = b_2, \\ d_3 x_3 + c_3 x_4 = b_3, \\ \dots \\ d_{n-1} x_{n-1} + c_{n-1} x_n = b_{n-1}, \\ d_n x_n = b_n, \end{cases} \quad (7)$$

and solutions are obtained by the following relations

$$\begin{aligned} x_n &= \frac{b_n}{d_n}, \\ x_i &= \frac{b_i - c_i x_{i+1}}{d_i}, \quad i = n-1, n-2, \dots, 1. \end{aligned} \quad (8)$$

In the pseudo C/C++ code (again remember that the indices run from 0 to n-1) we have

```

x[n-1] = b[n-1]/d[n-1];
for (i=n-2; i≥0; i--) x[i] = (b[i] - c[i]*x[i+1])/d[i];

```

Poisson's equation

Let $\Omega \subset \mathbb{R}^N$ be an open and bounded set. We are looking for a continuous function $u : \bar{\Omega} \rightarrow \mathbb{R}$ twice differentiable in Ω satisfying the conditions

$$\begin{cases} \Delta u = f & \text{in } \Omega, \\ u = g & \text{on } \partial\Omega. \end{cases} \quad (9)$$

where $f : \Omega \rightarrow \mathbb{R}$, $g : \partial\Omega \rightarrow \mathbb{R}$ are given continuous functions. The second relation in (9), i.e.

$u(x) = g(x)$ for $x \in \partial\Omega$ is the *boundary condition* which is called the *Dirichlet boundary condition*.

In a special case when $f = 0$ we have the Laplace equation $\Delta u = 0$ and the Dirichlet problem for Laplace equation is now

$$\begin{cases} \Delta u = 0 & \text{in } \Omega, \\ u = g & \text{on } \partial\Omega. \end{cases} \quad (10)$$

These equations appear frequently in physical sciences. For example, solutions to (9) correspond to steady state of time evolutions such as heat flow, matter flow (diffusion) or wave motion. The function f represents heat sources in case of the thermal phenomena.

Both these equation arise also in electrodynamics and gravitation theory, where electric, magnetic or gravitation fields are given as the gradient of the potential function. For example, suppose that E is the electric field and ρ is the distribution of charge in some region $\Omega \subset \mathbb{R}^3$ i.e.

$$\begin{aligned} E : \Omega &\rightarrow \mathbb{R}^3, \\ \rho : \Omega &\rightarrow \mathbb{R}, \end{aligned}$$

where the charge distribution ρ is given and electric field E is to be determined. The relation between both quantity is the first Maxwell equation (known as the Gauss law)

$$\operatorname{div} E = \frac{1}{\varepsilon_0} \rho, \quad (11)$$

where ε_0 is the vacuum permittivity (the electric constant), in SI units $\varepsilon_0 = 8.854 \cdot 10^{-12} \text{ C}^2 / (\text{N} \times \text{m}^2)$. In the case of static distribution of charge (the charge density does not depend on time, $\rho = \rho(x)$) the electric field is conservative and can be expressed by the gradient of the potential function $\psi(x)$

$$E = -\nabla \psi = -\left(\frac{\partial \psi}{\partial x_1}, \frac{\partial \psi}{\partial x_2}, \frac{\partial \psi}{\partial x_3} \right). \quad (12)$$

When we substitute (12) into (11) we get the Poisson equation

$$\Delta \psi = -\frac{1}{\varepsilon_0} \rho \quad \text{in } \Omega, \quad (13)$$

for the electric potential. In particular in the region where no charge is present the potential satisfies the Laplace equation $\Delta \psi = 0$.

We begin our study of numerical methods for the Poisson equation with the one dimensional case $\Omega = (a, b) \subset \mathbb{R}^1$, $u : [a, b] \rightarrow \mathbb{R}$. In this case the boundary consists just of two points $\partial\Omega = \{a, b\}$ so the boundary condition is simply expressed by two numbers $g_0, g_1 \in \mathbb{R}$. Thus our model problem reads

$$\begin{cases} -\frac{d^2 u}{dx^2} = f(x) & x \in (a, b), \\ u(a) = g_0, u(b) = g_1. \end{cases} \quad (14)$$

In this problem given are: continuous function $f : [a, b] \rightarrow \mathbb{R}$, boundary values $g_0, g_1 \in \mathbb{R}$. We are looking for the continuous function $u : [a, b] \rightarrow \mathbb{R}$ on the closed interval $[a, b]$ which is twice differentiable in the open interval (a, b) . Because the assumption is that $f \in C([a, b])$, so we can write that $u \in C([a, b]) \cap C^2((a, b))$.

Finite difference approximation

We introduce on the interval $[a, b]$ the uniform grid of points $\{x_i\}_{i=0}^{n+1}$ such that $x_{i+1} - x_i = h$. Thus the relation between $h > 0$ and $n \in \mathbb{N}$ is following: $h = \frac{b-a}{n+1}$. The approximation to the solution of (14) is a sequence $\{u_0, u_1, \dots, u_n, u_{n+1}\}$ which is meant to satisfy $u(x_i) \approx u_i$, where u is the true solution. The second derivative $\frac{d^2 u(x)}{dx^2}$ can be approximated by the three-points scheme with equal distances

$$\frac{d^2 u}{dx^2}(x) \approx \frac{u(x+h) - 2u(x) + u(x-h)}{h^2}. \quad (15)$$

In our notation $u_i \approx u(x_i)$, $u_{i+1} \approx u(x_{i+1}) = u(x_i + h)$, $u_{i-1} \approx u(x_{i-1}) = u(x_i - h)$, so the discretized form of the problem (14)

$$\begin{cases} -\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} = f(x_i), & i = 1, \dots, n, \\ u_0 = d_0, u_{n+1} = d_1. \end{cases} \quad (16)$$

Let us rewrite this system as

$$\begin{cases} 2u_1 - u_2 = h^2 f(x_1) + d_0, & i = 1, \\ -u_{i-1} + 2u_i - u_{i+1} = h^2 f(x_i), & i = 2, \dots, n-1, \\ 2u_n - u_{n-1} = h^2 f(x_n) + d_1, & i = n. \end{cases} \quad (17)$$

For better visualization let us present this system in an expanded matrix form

$$\begin{bmatrix} 2 & -1 & 0 & 0 & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 \\ 0 & -1 & 2 & -1 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & -1 & 2 & -1 \\ 0 & 0 & \dots & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} = \begin{bmatrix} h^2 f_1 + d_0 \\ h^2 f_2 \\ h^2 f_3 \\ \vdots \\ h^2 f_{n-1} \\ h^2 f_n + d_1 \end{bmatrix}, \quad (18)$$

which is tridiagonal – see (5). It has even simpler form than the general tridiagonal system (4), because the values on each diagonal are the same: -1 for below or above, and 2 for the main diagonal.

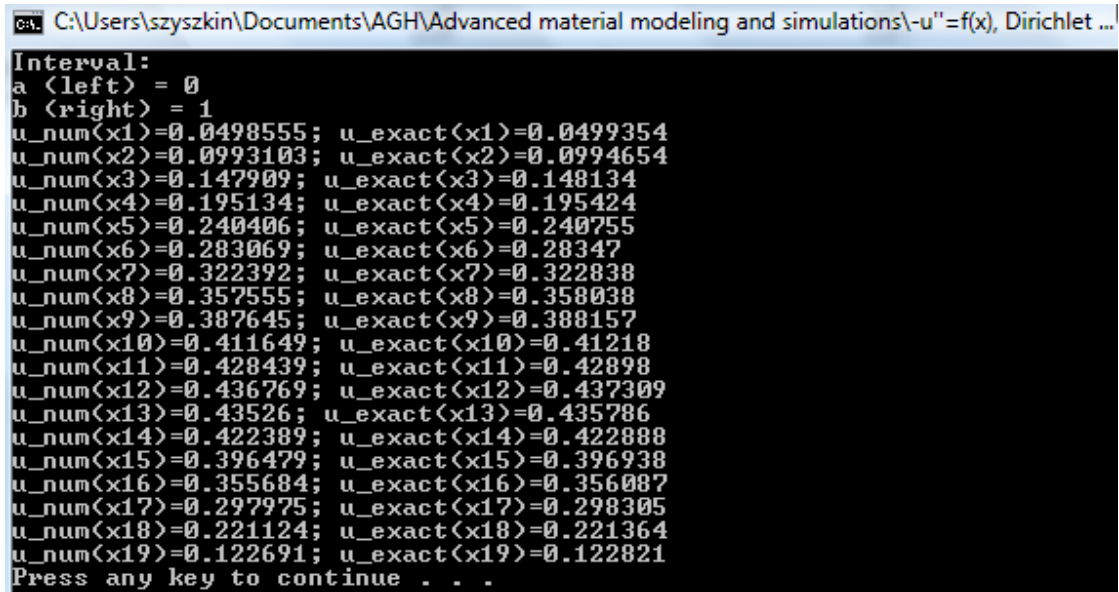
Example. Let us consider the following problem

$$\begin{cases} -\frac{d^2u}{dx^2} = x(3+x)e^x, & x \in (0, 1), \\ u(0) = 0, u(1) = 0. \end{cases}$$

One can easily check that the exact solution is given by

$$u_{exact}(x) = x(1-x)e^x.$$

When we run the program `-u''(x)=f(x), Dirichlet boundary.cpp` with the $n = 20$ results are following



```
C:\Users\szyszk\Documents\AGH\Advanced material modeling and simulations\'-u''=f(x), Dirichlet ...
Interval:
a <left> = 0
b <right> = 1
u_num<x1>=0.0498555; u_exact<x1>=0.0499354
u_num<x2>=0.0993103; u_exact<x2>=0.0994654
u_num<x3>=0.147909; u_exact<x3>=0.148134
u_num<x4>=0.195134; u_exact<x4>=0.195424
u_num<x5>=0.240406; u_exact<x5>=0.240755
u_num<x6>=0.283069; u_exact<x6>=0.28347
u_num<x7>=0.322392; u_exact<x7>=0.322838
u_num<x8>=0.357555; u_exact<x8>=0.358038
u_num<x9>=0.387645; u_exact<x9>=0.388157
u_num<x10>=0.411649; u_exact<x10>=0.41218
u_num<x11>=0.428439; u_exact<x11>=0.42898
u_num<x12>=0.436769; u_exact<x12>=0.437309
u_num<x13>=0.43526; u_exact<x13>=0.435786
u_num<x14>=0.422389; u_exact<x14>=0.422888
u_num<x15>=0.396479; u_exact<x15>=0.396938
u_num<x16>=0.355684; u_exact<x16>=0.356087
u_num<x17>=0.297975; u_exact<x17>=0.298305
u_num<x18>=0.221124; u_exact<x18>=0.221364
u_num<x19>=0.122691; u_exact<x19>=0.122821
Press any key to continue . . .
```

This figure shows about the numerical and exact solution of the boundary problem. Values for $x=0$ and $x=1$ are not included in the output. The total number of grid points is 21, the number of internal points is 19.