

Project 1

Due date **19.th of September, 2016 - 23:59**

Øyvind B. Svendsen, Magnus Christopher Bareid
un: oyvinbsv, magnucb

September 18, 2016

Abstract

The aim of this project is for the students to get familiar with various vector and matrix operations, from dynamic memory allocation to the usage of programs in the library package of the course.

The students were invited to use either brute force-algorithms to calculate linear algebra, or to use a set of recommended linear algebra packages, i.e. Armadillo, that simplify the syntax of linear algebra. Additionally, dynamic memory handling is expected.

The students will showcase necessary algebra to perform the tasks given to them, and explain the way said algebra is implemented into algorithms. In essence, we're asked to simplify a linear second-order differential equation from the form of the Poisson equation, seen as

$$\nabla^2 \Phi = -4\pi\rho(\mathbf{r})$$

into a one-dimensional form bounded by Dirichlet boundary conditions.

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

so that discretized linear algebra may be committed unto the equation, and yield a method for numerical integration.

Contents

1	Computational Physics, first project	3
1.1	The fundamental math	3
1.1.1	Introduction	3
1.1.2	Problem	3
1.1.3	Method	3

1 Computational Physics, first project

1.1 The fundamental math

1.1.1 Introduction

The production of this document will inevitably familiarize its authors with the programming language C++, and to this end mathematical groundwork must first be elaborated to translate a Poisson equation from continuous calculus form, into a discretized numerical form.

The Poisson equation is rewritten to a simplified form, for which a real solution is given, with which we will compare our numerical approximation to the real solution.

The real motivation for this project, though, is to find a viable numerical method for double integration and the implementation of such.

1.1.2 Problem

1.1.3 Method

Reviewing the Poisson equation:

$$\begin{aligned}\nabla^2 \Phi &= -4\pi\rho(\mathbf{r}), \text{ which is simplified one-dimensionally by } \Phi(r) = \phi(r)/r \\ \Rightarrow \frac{d^2\phi}{dr^2} &= -4\pi r\rho(r), \text{ which is further simplified by these substitutions:} \\ r &\rightarrow x, \\ \phi &\rightarrow u, \\ 4\pi r\rho(r) &\rightarrow f, \quad \text{which produces the simplified form}\end{aligned}$$

$$\begin{aligned}-u''(x) &= f(x), \quad \text{for which we assume that } f(x) = 100e^{-10x}, \\ \Rightarrow u(x) &= 1 - (1 - e^{-10})x - e^{-10x}, \text{ with bounds: } x \in [0, 1], \quad u(0) = u(1) = 0\end{aligned}\tag{1}$$

From here on and out, the methods for finding the doubly integrated function $u(x)$ numerically will be deduced.

To more easily comprehend the syntax from a programming viewpoint, one may refer to the each discretized representation of x and u ; we know the span of x , and therefore we may divide it up into appropriate chunks. Each of these x_i will yield a corresponding u_i .

We may calculate each discrete x_i by the form $x_i = ih$ in the interval from $x_0 = 0$ to $x_{n+1} = 1$ as it is linearly increasing, meaning we use $n + 1$ points in our approximation, yielding the step length $h = 1/(n + 1)$. Of course, this also yields the discretized representation of $u(x_i) = u_i$.

Through Euler's teachings on discretized numerical derivation methods, a second derivative may be constructed through the form of

$$\begin{aligned}\left(\frac{\partial}{\partial x}\right)_{fw} u(x) &= \frac{u_{i+1} - u_i}{h}, \quad \left(\frac{\partial}{\partial x}\right)_{bw} u(x) = \frac{u_i - u_{i-1}}{h}, \\ \left(\frac{\partial}{\partial x}\right)^2 u_i &= \left(\frac{\partial}{\partial x}\right)_{bw} \left(\frac{\partial}{\partial x}\right)_{fw} u_i = \left(\frac{\partial}{\partial x}\right)_{bw} \left(\frac{u_{i+1} - u_i}{h}\right) = \frac{\left(\frac{\partial}{\partial x}\right)_{bw} u_{i+1} - \left(\frac{\partial}{\partial x}\right)_{bw} u_i}{h} \\ \left(\frac{\partial}{\partial x}\right)^2 u_i &= \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}, \quad \text{which we then use for the problem in question, (1).} \\ \Rightarrow -\left(\frac{\partial}{\partial x}\right)^2 u(x) &= -\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} = \frac{2u_i - u_{i+1} - u_{i-1}}{h^2} = f_i, \quad \text{for } i = 1, \dots, n\end{aligned}\tag{2}$$

The discretized problem can now be solved as a linear algebra problem. Looking closer at the discretized problem:

$$\begin{aligned} -u''(x_i) &= \frac{-u_{i+1} + 2u_i - u_{i-1}}{h^2} = f_i, & \text{for } i = 1, \dots, n. \\ \Rightarrow & -u_{i+1} + 2u_i - u_{i-1} = h^2 f_i, & \text{substitute } h^2 f_i = y_i, \text{ and test for some values:} \end{aligned}$$

$$\begin{aligned} i = 1 : & \quad -u_2 + 2u_1 - u_0 = y_1 \\ i = 2 : & \quad -u_3 + 2u_2 - u_1 = y_2 \\ i = 3 : & \quad -u_4 + 2u_3 - u_2 = y_3 \\ & \quad \vdots \\ i = n : & \quad -u_{n+1} + 2u_n - u_{n-1} = y_n \end{aligned}$$

By now it should be obvious to recognize that the coefficients corresponding to each of these terms and their corresponding values of $u(x)$ looks very similar to a tridiagonal matrix multiplication problem which could be represented such as this:

$$-\left(\frac{\partial}{\partial x}\right)^2 u(x) = f(x) \quad \Rightarrow \quad \hat{\mathbf{A}}\hat{\mathbf{u}} = \hat{\mathbf{y}} \quad \Rightarrow \quad \begin{pmatrix} 2 & -1 & 0 & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & & \vdots \\ 0 & -1 & 2 & \ddots & \ddots & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & \ddots & -1 \\ 0 & \dots & \dots & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ u_{n+1} \end{pmatrix} = \begin{pmatrix} y_0 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ y_{n+1} \end{pmatrix}$$

This matrix equation will not be valid for the first, and last, values of $\hat{\mathbf{y}}$ because they would require elements of $\hat{\mathbf{u}}$ that are not defined; u_{-1} and u_{n+1} , respectively. Given this constraint we see that the matrix-equation gives the same set of equations that we require.

$$\begin{aligned} i = 1 : & \quad -u_2 + 2u_1 - u_0 = y_1 \\ i = 2 : & \quad -u_3 + 2u_2 - u_1 = y_2 \\ i = 3 : & \quad -u_4 + 2u_3 - u_2 = y_3 \\ & \quad \vdots \\ i = n : & \quad -u_{n+1} + 2u_n - u_{n-1} = y_n \end{aligned}$$

The original problem at hand (the simplified Poisson equation) has now been reduced to a numerical linear algebra problem. Solving a tridiagonal matrix-problem like this is done by Gaussian elimination of the tridiagonal matrix $\hat{\mathbf{A}}$, and thereby solving $\hat{\mathbf{u}}$ for the resulting diagonal-matrix, as presumably $\hat{\mathbf{A}}$ and $\hat{\mathbf{y}}$ are the knowns in this set.

Firstly the tridiagonal matrix $\hat{\mathbf{A}}$ is rewritten to a series of three vectors $\hat{\mathbf{a}}$, $\hat{\mathbf{b}}$, and $\hat{\mathbf{c}}$ that will represent a general tridiagonal matrix. This will make it easier to include other problems of a general form later.

The tridiagonal matrix $\hat{\mathbf{A}}$ with the vector $\hat{\mathbf{y}}$ included now looks like:

$$\begin{pmatrix} b_0 & c_0 & 0 & \dots & \dots & 0 & y_0 \\ a_1 & b_1 & c_1 & \ddots & & \vdots & \vdots \\ 0 & a_2 & b_2 & c_2 & \ddots & \vdots & \vdots \\ \vdots & \ddots & a_3 & b_3 & \ddots & 0 & \vdots \\ \vdots & & \ddots & \ddots & \ddots & c_n & \vdots \\ 0 & \dots & \dots & 0 & a_{n+1} & b_{n+1} & y_{n+1} \end{pmatrix}$$

, but we only work with rows from $i = 1$ to $i = n$ because of the Dirichlet conditions, as explained above. The matrix which we row-reduce thus looks like this

$$\begin{pmatrix} b_1 & c_1 & 0 & \dots & \dots & 0 & y_1 \\ a_2 & b_2 & c_2 & \ddots & & \vdots & \vdots \\ 0 & a_3 & b_3 & c_3 & \ddots & \vdots & \vdots \\ \vdots & \ddots & a_4 & b_4 & \ddots & 0 & \vdots \\ \vdots & & \ddots & \ddots & \ddots & c_n & \vdots \\ 0 & \dots & \dots & 0 & a_n & b_n & y_n \end{pmatrix}$$

The Gaussian elimination can be split into two parts; a forward substitution where the matrix-elements a_i are set to zero, and a backward substitution where the vector-elements u_i are calculated from known values.

Starting the Gaussian elimination with the second row, row II, a row operation is performed to maintain the validity of the system. The goal is to remove element a_2 from the row. This is done by subtracting row II, multiplied with some constant k from row I. For every next row operation, there will then be a new k_i calculated.

$$\begin{pmatrix} b_1 & c_1 & 0 & \dots & \dots & 0 & y_1 \\ \tilde{a}_2 & \tilde{b}_2 & \tilde{c}_2 & \ddots & & \vdots & \tilde{y}_2 \\ 0 & a_3 & b_3 & c_3 & \ddots & \vdots & \vdots \\ \vdots & \ddots & a_4 & b_4 & \ddots & 0 & \vdots \\ \vdots & & \ddots & \ddots & \ddots & c_n & \vdots \\ 0 & \dots & \dots & 0 & a_n & b_n & y_n \end{pmatrix}$$

$$\tilde{\Pi} = \Pi - k_I \times \text{I}$$

where k_I is determined by

$$\tilde{a}_2 = 0 = a_2 - k_I b_1 \Rightarrow k_I = \frac{a_2}{b_1}$$

$$\tilde{b}_2 = b_2 - \frac{a_2}{b_1} c_1$$

$$\tilde{c}_2 = c_2 - \frac{a_2}{b_1} \times 0 = c_2$$

$$\tilde{y}_2 = y_2 - \frac{a_2}{b_1} y_1$$

Moving on to row 3, and performing a similar operation:

$$\begin{pmatrix} b_1 & c_1 & 0 & \dots & \dots & 0 & y_1 \\ 0 & \tilde{b}_2 & c_2 & \ddots & & \vdots & \tilde{y}_2 \\ 0 & \tilde{a}_3 & \tilde{b}_3 & \tilde{c}_3 & \ddots & \vdots & \tilde{y}_3 \\ \vdots & \ddots & a_4 & b_4 & \ddots & 0 & \vdots \\ \vdots & & \ddots & \ddots & \ddots & c_n & \vdots \\ 0 & \dots & \dots & 0 & a_n & b_n & y_n \end{pmatrix}$$

$$\begin{aligned}
& \tilde{\text{III}} = \text{II} - k_{\text{II}} \times \text{II} \\
& \text{where } k_{\text{II}} \text{ is determined by} \\
& \tilde{a}_3 = 0 = a_3 - k_{\text{II}} \tilde{b}_2 \quad \Rightarrow k_{\text{II}} = \frac{a_3}{\tilde{b}_2} \\
& \tilde{b}_3 = b_3 - \frac{a_3}{\tilde{b}_2} c_2 \\
& \tilde{c}_3 = c_3 - \frac{a_3}{\tilde{b}_2} \times 0 = c_3 \\
& \tilde{y}_3 = y_3 - \frac{a_3}{\tilde{b}_2} \tilde{y}_2
\end{aligned}$$

Having repeated this procedure, a pattern emerges and an algorithm can be formulated:

$$\begin{aligned}
\tilde{b}_{i+1} &= b_{i+1} - \frac{a_{i+1}}{\tilde{b}_i} c_i \\
\tilde{y}_{i+1} &= y_{i+1} - \frac{a_{i+1}}{\tilde{b}_i} \tilde{y}_i \\
i &= 1, 2, \dots, n-1
\end{aligned}$$

After this procedure, the tridiagonal matrix $\hat{\mathbf{A}}$ is transformed into an uppertriangular matrix. This sort of set of equations can be solved iteratively for $\hat{\mathbf{u}}$, since the last equation has one unknown and the other equations has only two unknowns.