Abstract

We present our Ferrari algorithm for solving linear equations. Our best algorithm runs as $4n$ FLOPS with n the dimensionality of the matrix.

Project 1

Johanne Mehren, Stine Sagen and Marit Kollstuen

September 6, 2018

1 Introduction

The main goal of project 1 is to understand the concept of dynamic memory allocation, a memory handling frequently used in programs such as C++ and Fortran. In C++ you have three ways of managing memory; statically, locally or dynamically. By using dynamic memory allocation, we are reserving space for saving data and are able to manage the lifetime of allocated memory.

2 Theory, algorithms and methods

2.1 Exact solution

The one-dimensional Poisson equation

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

does have an exact (analytical) solution of the form:

$$u(x) = 1 - (1 - e^{-10})x - e^{-10x} \tag{2}$$

when assuming the source term on the right hand side of the Poisson equation is

$$f(x) = 100e^{-10x}. \tag{3}$$

Substituting the above solution into our differential equation, we can then verify that this turns out to be correct.

$$\begin{aligned} u(x) &= 1 - (1 - e^{-10})x - e^{-10x} \\ u'(x) &= -1 + e^{-10} + 10e^{-10x} \\ u''(x) &= -100e^{-10x} \\ -(-100e^{-10x}) &= 100e^{-10x} \\ &\therefore \\ 100e^{-10x} &= 100e^{-10x} \end{aligned}$$

As expected, $u(x) = 1 - (1 - e^{-10})x - e^{-10x}$ is an exact solution of $-u''(x) = f(x)$.

Our differential equation conserns a boundary value problem which means our solution also needs to satisfy the given boundary conditions.
Checking the behavior of the exact solution at its bondary points 0 and 1:

$$u(0) = 1 - (1 - e^{-10}) \cdot 0 - e^{-10 \cdot 0} = 0$$
$$u(1) = 1 - (1 - e^{-10}) \cdot 1 - e^{-10 \cdot 1} = 0$$

$u(x) = 1 - (1 - e^{-10})x - e^{-10x}$ is indeed a solution of our boundary value problem.

In our project we want to compare this exact solution with the numerical solution we obtain when the boundary value problem takes a discretized form.

2.2 Finite difference method

We want to solve the Poisson equation numerically and this can be achived by something we call finite-difference methods. Finite-difference methods are about replacing the derivatives appearing in the differential equation by finite difference approximations at a given set of discrete points in space and/or time. E.g when this method is applied to the second order derivative in the poisson equation, we obtain a dicretized approximation for u in the x-direction because its only depended on the x variable. In deriving these finite difference approximations, Taylor series expansion might be very useful.

2.2.1 Taylor series expansion

We might use Taylor series expansion in order to derive a suitable finite difference approximation for the differential equation in the process for obtaining a numerical solution.
Assuming our function u(x) is higher order differensible we can preform a Taylor expansion forward and backward in space:

$$u(x+h) = u(x) + hu' + \frac{h^2 u''}{2!} + \frac{h^3 u'''}{3!} + o(h^4)$$
$$u(x-h) = u(x) - hu' + \frac{h^2 u''}{2!} - \frac{h^3 u'''}{3!} + o(h^4)$$

To obtain finite difference approximation for the second order derivative we then sum the two equations:

$$u(x+h) + u(x-h) = 2u(x) + \frac{2h^2 u''}{2!} + o(h^4)$$

In the end we solve for u'' and get:

$$u'' = \frac{u(x+h) + u(x-h) - 2u(x)}{h^2} + o(h^2)$$

2.2.2 Discretization of the bondary value problem

As already mentioned the concept of discretizing is about dividing the domain into a finite set of discrete points. We now the discretize u as v_i with grid points $x_i = ih$ in the range from $x_0 = 0$ to $x_{n+1} = 1$. The grid spacing is defined as $h = \frac{x_{n+1}-x_0}{n+1} = \frac{1-0}{n+1} = \frac{1}{n+1}$. The boundary conditions are $v_0 = v_{n+1} = 0$. Our approximated finite difference approximation for the poisson equation is now:

$$-\frac{v_{i+1} + v_{i-1} - 2v_i}{h^2} \tag{4}$$

2.3 A general algorithm for solving the tridiagonal matrix

2.4 Specialized algorithm for solving the tridiagonal matrix

2.5 Relative error

2.6 LU-decomposition

3 Project 1 a)

We are attempting to solve the equation:

$$-u''(x) = f(x), x \in (0,1), u(0) = u(1) = 0$$

which can be approximated as:

$$v_i" \approx -\frac{v_{i-1} - 2v_i + v_{i+1}}{h^2} \tag{5}$$

Assumed $n = 4$, it can be shown that this can be represented as a Toepliz-matrix by setting:

$$\begin{bmatrix} v \end{bmatrix} = \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix}$$

For which

$$\begin{bmatrix} v_1" \\ v_2" \\ v_3" \\ v_4" \end{bmatrix} \approx -\frac{1}{h^2} \begin{bmatrix} -(v_0 - 2v_1 + v_2) \\ -(v_1 - 2v_2 + v_3) \\ -(v_2 - 2v_3 + v_4) \\ -(v_3 - 2v_4 + v_5) \end{bmatrix}$$

The boundary conditions are set to $v_0 = v_{n+1} = 0$, so the expression becomes:

$$\begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = -h^2 \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix}$$

In which f_i is known.

4 Project 1 b)

5 Project 1 c)

In this task we implement matrix

6 Project 1 d)

7 Project 1 e)

8 Conclusion

9 References