Project 1: Vector and Matrix Operations in c++, Fys 4150

Ask J. Markestad, Thorbjørn V. Larsen, Ingrid A. V. Holm

Introduction

Differential equations are fundamental in physics. The standard approach in most branches of physics is to describe a physical system in terms of it's symmetries and degrees of freedom through the formalism of Lagrange or Hamilton, i.e. by setting up equations of motion. It is thus a very important tool to be able to solve these differential equations as precisely and effectively as possible. In this project we explore algorithms for calculating second order differential equations using vector and matrix operations with the aim to better understand the computational demands of differential equation solutions and how to reduce these demands.

Theory and Algorithms

We will be looking in detail at linear second-order differential equations.

$$\frac{d^2y}{dx^2} = k^2(x)y = f(x)$$

More specifically, we will look at differential equations where $k^2(x) = 0$. A known physics example of this type of equation would be Poisson's equation in radial coordinates for spherically symmetric source term.

$$\frac{d^2\phi}{dr^2} = f(r)$$

In Newtonian gravity or in electromagnetism the source term is associated with a charge or mass distribution with a negative sign to explicitly indicate the attractiveness of the resulting potential. We will therefore look closely at the equation:

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

With Dirichlet boundary conditions, i.e. $x \in (0,1)$, $u(0) = u(1) = 0$. To solve this equation numerically we need to discretize the functions u and f with grid points $x_i = ih$ from 0 to 1 in $n+1$ steps using step length $h = 1/(n+1)$. This means that we know the initial and final point of u through the boundary conditions, and that we can use the three point derivative formula to rewrite the equation as:

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

The solution to this equation with the given boundary conditions is $u(x) = 1 - (1 - e^{-10}) - e^{-10x}$. For $x = 0$ we get $u(0) = 1-(1-e^{-10})*0-e^{-10*0} = 1-1 = 0$, and for $x = 1$ we get $u(1) = 1-1+e^{-10}-e^{-10} = 0$, so it satisfies the boundary conditions and by deriving it twice we get $u'(x) = -(1-e^{-10})+10e^{-10x}$, $u''(x) = -100e^{-10x}$. So we have found the exact solution to the differential equation which gives us a value to compare to, to see the accuracy of our numerical method. The next step to solve the equation is to rewrite it as a linear set of equations:

$$\begin{pmatrix} 2 & -1 & 0 & ... & ... & 0 \\ -1 & 2 & -1 & 0 & ... & 0 \\ 0 & -1 & 2 & -1 & 0 & ... \\ ... & ... & ... & ... & ... & ... \\ 0 & ... & 0 & -1 & 2 & -1 \\ 0 & 0 & ... & 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ ... \\ ... \\ u_n \end{pmatrix} = \begin{pmatrix} f_1/h^2 \\ f_2/h^2 \\ f_3/h^2 \\ ... \\ ... \\ f_n/h^2 \end{pmatrix} \tag{3}$$

We will now take a step back and look at a more general matrix **A** for this sort of linear problem and developing an algorithm for it. This allows us to see the difference in CPU time between the general algorithm and one made specifically for our very symmetric case. The more general matrix we will look at is a tridiagonal matrix:

$$\mathbf{A} = \begin{pmatrix} b_1 & c_1 & 0 & ... & ... & 0 \\ a_1 & b_2 & c_2 & 0 & ... & 0 \\ 0 & a_2 & b_3 & c_3 & 0 & ... \\ ... & ... & ... & ... & ... & ... \\ 0 & ... & 0 & a_{n-2} & b_{n-1} & c_{n-1} \\ 0 & 0 & ... & 0 & a_{n-1} & b_n \end{pmatrix} \tag{4}$$

Results

Conclusion

References

[1] Name of Author/s *Name of work* Prublisher year

`Web link`