ReportProject 1

# Introduction

In this project the aim is to explore different numerical methods for lowering computation times and memory loads while maintaining precision. The Poisson equation is solved with Dirichlet boundary conditions by rewriting it as a set of linear equations, and then the aforementioned exploration of numerical methods is applied on these linear equations.

# Theory

## The Poisson equation

The following equation will be solved:

## A simple algorithm for solving tridiagonal matrices

With a 2nd order Euler approximation a differential equation can be converted into a tridiagonal matrix.

Dette tilsvarer matrisen:

The resulting matrix can be represented by the three vectors a, b and c, which is beneficial if we want to store the matrix on our computer. This way the memory requirement would be bits rather than bits. Reducing the required ram storage opens the possibility of rather than approximately on a standard home computer.

Because of the nature of the diagonal matrix we can now use the simple following algorithm solve for the second derivative:

With the intent of performing we first remove the a’s with the following row operations on matrix A:

Likewise, we substitute back to remove the c’s

The resulting matrix then looks like this:

With the final

Insert math stuff here.

## Point about flops

Just counting the flops and runtime here

## Error

Because a 2nd order Euler approximation is used we expect to find an error no greater than the third link in the Euler equation chain. If we call the error , this means that . However, we do have another possible source of error, and that is the computational error. When performing several million loops it is quite possible to lose some accuracy due to round off errors, in the sense that a float only stores so many digits. If n becomes too large the errors may stack up, resulting in potentially sporadic and catastrophic errors.

A float can store a total of (insert number here) digits, something something.

## Something about LU-decomposition

# Results

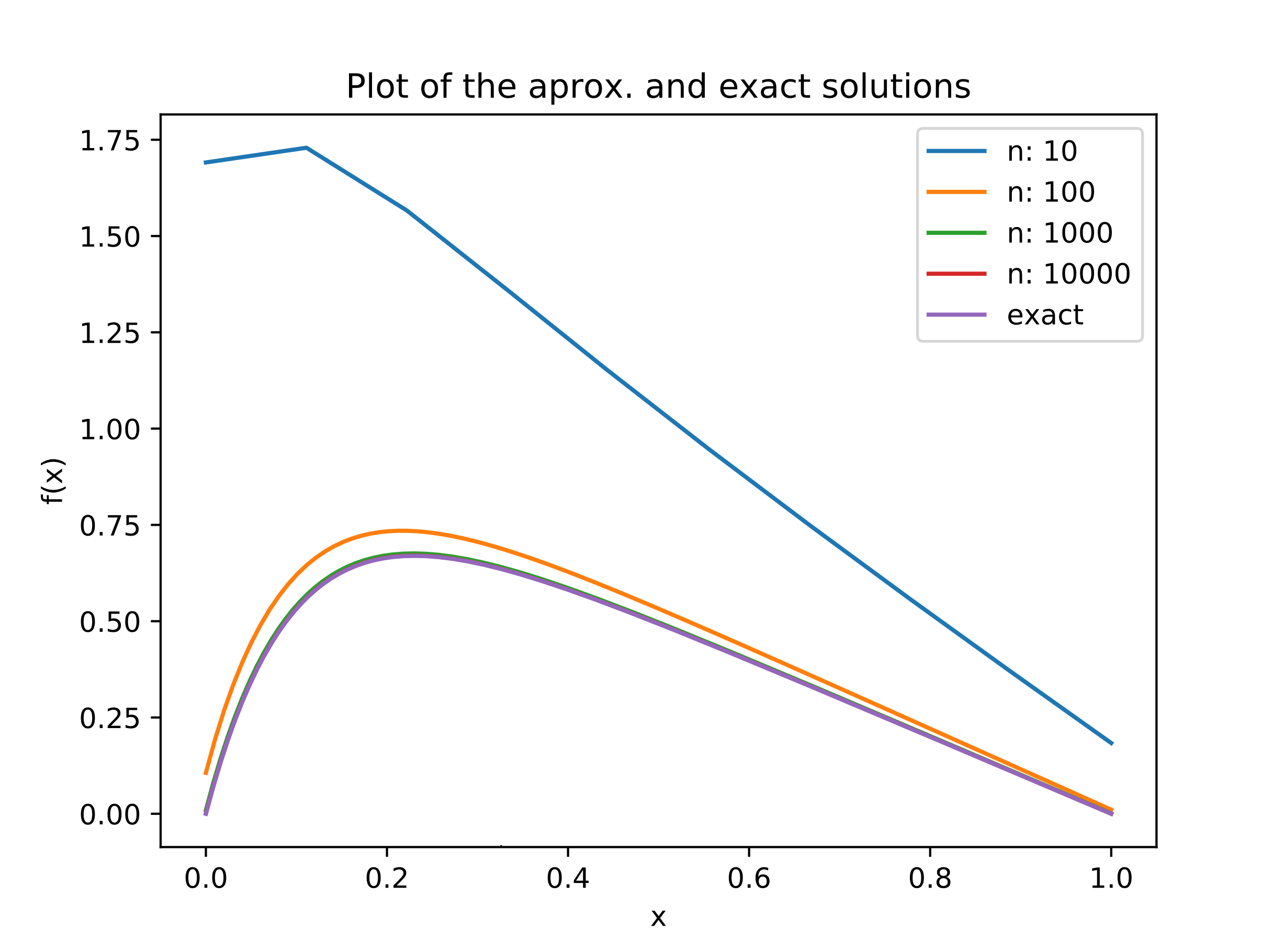


Figure 1 The numerical solution rapidly approaches the exact solution as n increases.

Something about flops and runtime

Something about LU-decomposition

# Conclusions

By tailoring the algorithm to the specific equation, the memory requirements are reduced from to , and some other stuff.

# References

There are none