Study of multiple Heat Diffusion schemes through a C++ implementation

Yoann Masson
Student in Software Engineering at Cranfield University

5th of december, 2017

Abstract

This report is about studying 4 schemes to implement a numerical solution of the heat diffusion equation in a 1D material through time. The equation is given by the following formula:

$$\frac{\partial T}{\partial t} = D\frac{\partial^2 T}{\partial x^2} \tag{1}$$

The constraint are the following, a one foot long wall at an initial temperature of 100°F is being subject to a rise of its surface temperature to 300°F. The diffusivity D of the material is 0.1ft^2/hr. We will study the rise of the temperature of the material with time t = 0.0, 0.1, 0.2, 0.3, 0.4, 0.5.
We will see four different schemes, each of them is providing a numerical solution. Those schemes are split into two categories:

- explicit schemes, schemes where there is only one unknown in the equation making it eazier to implement but less acurate
  - DuFort & Frankel Method
  - Richardson Method
- implicit schemes, schemes where there are mutiple unkowns that needs to be solved through a linear system, which is harder and more time consuming to solve, but are far more accurate.
  - Laasonen Method
  - Crank & Nicolson Method

The four methods will give four different solutions that we will compare with the known analytical solution. Firstly the discussion will be oriented toward the behaviour of the errors and secondly we will discuss about the C++ solution that I provided.

Contents

Chapter 1

The analytical solution

1.1 The expected behaviour

By chance, the heat diffusion equation has been solved and we know the temperature at the given time t and space x, with the given formula:

$$T = T_{sur} + 2(T_{in} - T_{sur}) \sum_{1}^{\infty} e^{-D(m\pi/L)^2 t} \frac{1-(-1)^m}{m\pi} sin(\frac{m\pi x}{L}) \tag{1.1}$$

If we plot the result for the wanted times, which are from time = 0 to time = 0.5 with a 0.1 time step we get the following plot:

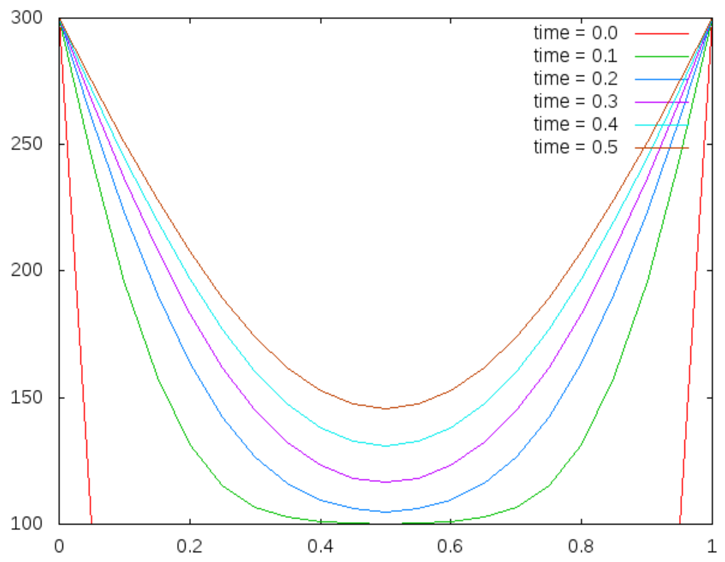

Figure 1.1: Analytical solution

As expected as the time flows the temperature is rising from the surface to the center. This is usefull to understand the heat diffusion behaviour. We will later use the data, used to make this plot, to compare with the other solution. This plot has been computed with a m value of 50, which we will discuss in the next section.

1.2 Study of m

$$\sum_{1}^{\infty} e^{-D(m\pi/L)^2 t} \frac{1-(-1)^m}{m\pi} sin(\frac{m\pi x}{L}) \tag{1.2}$$

When considering the analytical equation of the heat diffusion, we can see that there is a sum to the infinty involving the term m, which is of course not computable since we do not have an infinit time/space ressources to compute. In computer science, we can consider stopping the computation of the sum if the steps are becoming smaller and smaller, to the point where it is not use anymore to continue computing steps.

In this case, we are dealing with numbers in a range of 100 to 300, so I considered that an error less than 10^{-4} is acceptable. To know when the m value is to be stopped I printed out the same time and space with different m value. Time is t = 0.01,0.25,0.5 and space is x = 0.5.

m value	temperature at t = 0.5	temperature at t = 0.25	temperature at t = 0.01
1	144.5380	101.0325	47.8530
5	145.5377	110.1387	85.7275
10	145.5377	110.1389	95.4361
20	145.5377	110.1389	100.1123
49	145.5377	110.1389	100.0000
50	145.5377	110.1389	100.0000
100	145.5377	110.1389	100.0000

Table 1.1: Temperature by m value

The outcome of those results is that in this example, we don't need such a high value for m, we can see that values stop varying after m ¿ 49. If we want an error less than 10^{-4}, m = 50 is enough. Thanks to that, we know that we don't have to waste computer ressources and time on computing the analytical solution with a higher m value.

Chapter 2

Laasonen method, time step size and computation time

2.1 Theory

Laasonen method The Laasonen method is a way of solving the heat diffusion equation by computing a result step by step. The Laasonen method is part of the implicit scheme meaning that each time step not only relies on the previous one but also one the current one. When computing the result, the previous time step has already been computed but the current one will need to be solved through a linear system. Let's see the equation to get a clearer idea:

$$\frac{\partial T}{\partial t} = D\frac{\partial^2 T}{\partial x^2} \tag{2.1}$$

$$\frac{f\binom{n+1}{i} - f\binom{n}{i}}{\Delta t} = D\frac{f\binom{n+1}{i+1} - 2f\binom{n+1}{i} + f\binom{n+1}{i-1}}{\Delta x^2} \tag{2.2}$$

$$f\begin{pmatrix}n\\ i\end{pmatrix} = -Cf\begin{pmatrix}n+1\\ i+1\end{pmatrix} + (1+2C)f\begin{pmatrix}n+1\\ i\end{pmatrix} - Cf\begin{pmatrix}n+1\\ i-1\end{pmatrix} \text{ with } C = D\frac{\Delta t}{\Delta x^2} \tag{2.3}$$

$$\begin{pmatrix}1+2C & -C & 0 & ... & 0\\ -C & 1+2C & -C & ... & 0\\ ... & ... & ... & ... & ...\\ 0 & ... & -C & 1+2C & -C\\ 0 & ... & 0 & -C & 1+2C\end{pmatrix}\begin{pmatrix}f\binom{n+1}{0}\\ f\binom{n+1}{1}\\ ...\\ ...\\ f\binom{n+1}{k}\end{pmatrix} = \begin{pmatrix}f\binom{n}{0}\\ f\binom{n}{1}\\ ...\\ ...\\ f\binom{n}{k}\end{pmatrix} \tag{2.4}$$

With the above equations, we can see that we need to solve a linear system for each time step. Linear system can be really time and ressources consum-

ming for a computer. That's why we need a good algorithm to resolve this linear system, if we take a look at the left matrix we can see that it is a diagonal matrix.
A linear system with a diagonal matrix can be solved with Thomas' algorithm, which is a really good linear system solving algorithm that works with diagonal matrix, it's time complexity is of: *O(2n)* with n being the size of the matrix, so the number of space step we want, L=1 Δt=0.05 so we have 21 space steps. Meaning that a matrix twice larger will "only" take four time the requiered time. (A bit more explanation about Thomas' algorithm in the Appendix).

Since the method relies on computing different time with a constant time step, a good interrogation would be: what is a good time step ? In theory, considering a stable method we can expect a more accurate solution the smaller the time step is.

2.2 Pratice

Having a smaller and smaller difference between time steps allows us to have a better precision on each row, but if we decrease the time step we increase the number of linear system we need to solve. So I ran 4 simulations with various value of Δt, to compare the results. Let's see the norms of the errors matrices, the errors matrices is the Laasonen resulting matrix minus the analytical matrix:

Δt	Number of time step	One norm	Two norm	uniform norm	computing time
0.1	5	50.26	67.01	154.56	0.14ms
0.05	10	58.62	60.62	102.73	0.6ms
0.025	20	67.90	52.13	64.04	0.42ms
0.01	50	77.01	39.68	36.26	2.2ms

Table 2.1: Error norms according to Δt

Increasing the number of rows in the matrix by decreasing the time step size affects the one and two norms. Because the one norm is the highest error among the sums of error in each column and the two norm is just the sum of all errors. So more rows equals to possibly a worst one and two norms

one norm The one norm is about errors in one column, so for Δt = 0.1, we can say that on average the "worst" column has an error of 50.26/5=10.05. Considering the fact that we are dealing with numbers in range of 100-300, this is almost a 10% error on average. Compared to the Δt = 0.01 that has an average of 77.01/50 = 1.5 error, that's about 1%error.
According to the one norm, decreazing the size of time steps is a good idea.

two norm The two norm is just the square root of the sum errors, so increasing the numbers of rows in the matrix should increase the two norm, which is not the case. Between Δt = 0.1 and Δt = 0.05, we double the matrix size but the sum of errors is even smaller meaning that we have fewer errors overall.

uniform norm The uniform norm is the highest number among the sum of rows. Increasing the number of rows does not affect the behaviour of this number, so if this decreases with a smaller time step size, it means that the method is more and more accurate. By comparing the worst uniform norm 154.56 and the best 36.26, we can say that the worst row with a Δt = 0.1 is nearly six times less accurate than the worst row with a Δt = 0.01.

On a stable method such as Laasonen we have clearly seen that having a smaller Δt is giving the method a better accuracy, going from 10% to nearly 1% error with the first norm. In an ideal case we should find the best deal between time computing and errors, not to forget that If we want to be very accurate we would hurt ourself against errors implied by the numerical 64-bit system.

Chapter 3

Methods Result

3.1 Unstable Method

As we said already, thoses numerical schemes are based on computing steps after steps using the principe of derivatives. That is why, the more steps we have the more accurate results we have because the Δ in $f(x+\Delta x)$is smaller. This is introducing errors and we must be sure that the error are not comulating. In order to do this, we study the stability of the methods on the given problem. This means that every method does not work on every problem. Methods can be stable under certains conditions mostly relying on a Δx-Δt ratio. They can also be unconditionally stable/unstable meaning that for any Δx-Δt, the method will (or will not) work at any case.

Richardson case Let's take a look at Richardson method, it consists in derivating centrally the time and space.

$$\frac{\partial T}{\partial t} = D\frac{\partial^2 T}{\partial x^2} \tag{3.1}$$

$$\frac{f\binom{n+1}{i} - f\binom{n-1}{i}}{2\Delta t} = D\frac{f\binom{n}{i+1} - 2f\binom{n}{i} + f\binom{n}{i}}{\Delta x^2} \tag{3.2}$$

$$f\binom{n+1}{i} = Cf\binom{n}{i+1} - C2f\binom{n}{i} + Cf\binom{n}{i-1}) + f\binom{n-1}{i} \text{ with } C = 2D\frac{\Delta t}{\Delta x^2} \tag{3.3}$$

Now that we have only one unknown we can figure it out. Since Richardson's method is unstable (see Appendix for proof) considering this problem, the results are not coherent at all.

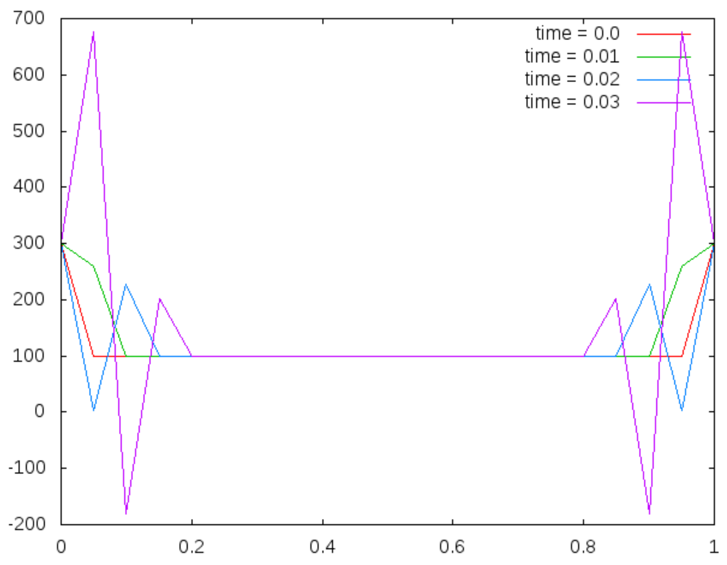

Figure 3.1: Richardson solution

I only plotted the three first steps because it shows that with only three steps we can clearly see that the methods is incoherent. Temperatures are supposed to go in a range of 100°F to 300°F. Which is clearly not the case here.

3.2 Stable Methods

Within the conditions defined in the original problem, Dufort-Frankel Laasonen and Crank-Nicholson methods are stable, which means that we do not have to fear for the errors to grow exponentially. We can even expected a better accuracy with time.

Mathematic The derivation of the original equation into the equations used in the schemes can be found in the appendix.

results The best way to compare the results are to compare the norms of their error matrix. For that, we can take a look at table 3.1 that compares the one norm , two norm, uniform norm and the computation time of the three methods.

Looking at those results, we can clearly see the superiority of the implicit methods (Crank-Nicolson & Laasonen) over the Dufort-Frankel method.

Method	One norm	Two norm	Uniform norm	Computation time
Crank Nicolson	14.95	14.48	20.77	2.04ms
Laasonen	77.01	39.68	36.26	3.17ms
Dufort-Frankel	85.61	74.47	82.82	0.93ms

Table 3.1: Accuracy of stable methods

The norms in this table represents the sum of errors, so the fewer errors the best a method is, regarding the considered norm. However, the computation time for Dufort-Frankel is far better. Crank-Nicolson seems to be the better deal here: the sum of the error of the least accurate time step is 20°F. So considering 21 space step, that's a 1°F error by number in average, which is acceptable. The second norm of Crank-Nicolson (14.48) that somehow represents the error in the all result matrix is also the best.
The outcome of this table is that if we have enough time we should consider using Crank-Nicolson since it is the more accurate method but if if we don't have enough computation time we should consider using Dufort Frankel. Dufort Frankel is a bit less accurate but is way faster because its computation only requires solving one equation and not a linear system.

Chapter 4

Implementation

Now that's we saw the theory and the results, let's take a look at the C++ implementation and the design of the application.

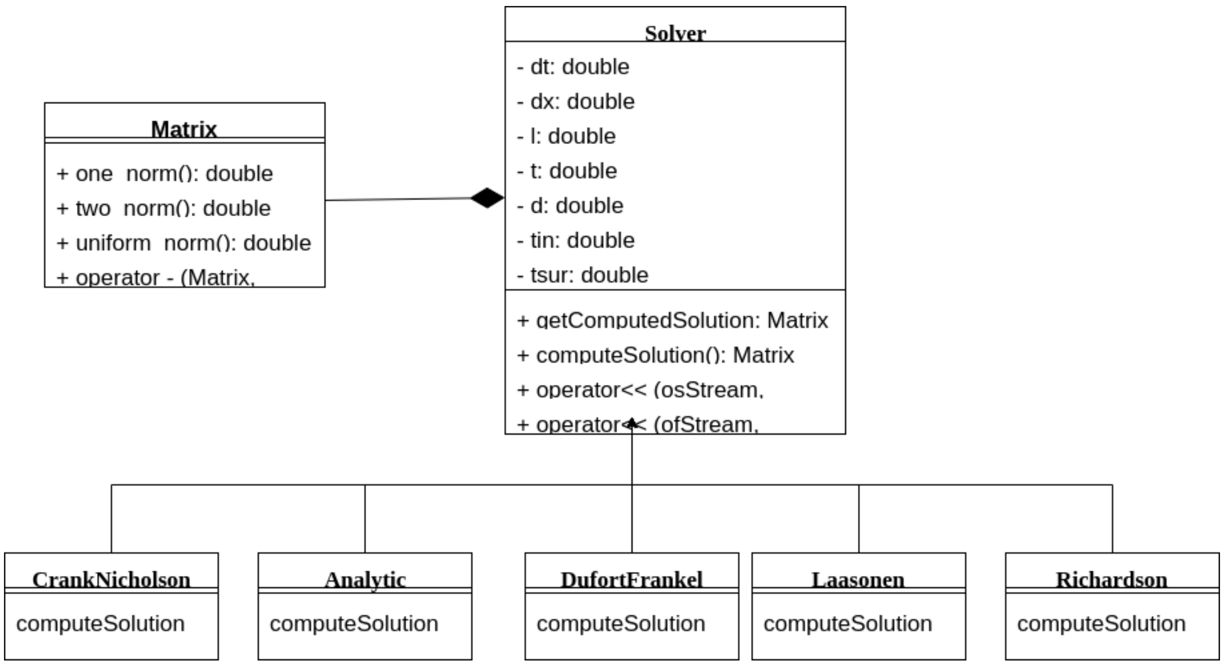

Figure 4.1: Class Diagramm

Class Diagram This is the class diagram of the application implementing the four schemes and the analytic solution. The*Solver* class uses *Matrix* to store the solution of the problem where rows are time steps and columns are space steps. *Solver* declares the virtual method *computeSolution()* so that

classes that inherit from it need to give an implementation of the solution for the child class to be instanciated in a *main* programm.
The base class *Solver* class holds all of the informations regarding the initial problem. Since it is attribut and not just hard-coded value in the code, all the child class are bounded to those value. Let's say that we want to see the solution with a surface temperature of 250°F instead of 300°F, we just have to instanciate the object with a different *Tsur* value.

functions To print out the solution both in screen or in a file, I added the redefinition of the " << " operator. On screen, the matrix will print the solution value for time [0.1, 0.2, 0.3, 0.4, 0.5] and will write on a file in a way that would be exploitable by GNUPlot. I added the minus operator in the matrix class to be able to substract matrices in order to get the error matrix. When instanciated an object of a Solver child class, you have to respect some rules in order not to get exceptions. For example, it is impossible to have a *dx* larger than a *L* it would not make any sense, so the programm would raise an *invalid_argument* exception

Conclusion

We had a problem that requiered a numerical approximation and four methods that would provide differents solutions with differents computation times. As we have seen the first thing to look at, is whther the method is stable or not to be sure to get "correct results". An other important aspect is the accuracy that we want to have, often the more accurate is a method, the more time consuming it will be, so we should look at the complexity of the method to know wether we can compute it in a reasonable time. Because all of the methods are numerical approch there will always be errors either implied by the method or by the limit of the machine, not being able to store the correct value of a number.
In the scope of the valid solutions, there is not a better solution above the others for all cases, it all depends on the computation time and on the accuracy wanted.

Appendix A

Mathematic derivation

Richardson

$$\frac{\partial T}{\partial t} = D\frac{\partial^2 T}{\partial x^2} \tag{A.1}$$

$$\frac{f\binom{n+1}{i} - f\binom{n-1}{i}}{2\Delta t} = D\frac{f\binom{n}{i+1} - 2f\binom{n}{i} + f\binom{n}{i}}{\Delta x^2} \tag{A.2}$$

$$f\begin{pmatrix}n+1\\ i\end{pmatrix} = Cf\begin{pmatrix}n\\ i+1\end{pmatrix} - C2f\begin{pmatrix}n\\ i\end{pmatrix} + Cf\begin{pmatrix}n\\ i-1\end{pmatrix}) + f\begin{pmatrix}n-1\\ i\end{pmatrix} \text{ with } C = 2D\frac{\Delta t}{\Delta x^2} \tag{A.3}$$

Dufort-Frankel

$$\frac{\partial T}{\partial t} = D\frac{\partial^2 T}{\partial x^2} \tag{A.4}$$

$$\frac{f\binom{n+1}{i} - f\binom{n-1}{i}}{2\Delta t} = D\frac{f\binom{n}{i+1} - (f\binom{n+1}{i} + f\binom{n-1}{i})) + f\binom{n}{i-1}}{\Delta x^2} \tag{A.5}$$

$$f\begin{pmatrix}n+1\\ i\end{pmatrix} = \frac{f\binom{n-1}{i} + 2C(f\binom{n}{i+1} - f\binom{n-1}{i} + f\binom{n}{i-1}))}{(1+2C)} \text{ with } C = 2D\frac{\Delta t}{\Delta x^2} \tag{A.6}$$

Laasonen

$$\frac{\partial T}{\partial t} = D\frac{\partial^2 T}{\partial x^2} \tag{A.7}$$

$$\frac{f\binom{n+1}{i} - f\binom{n}{i}}{\Delta t} = D\frac{f\binom{n+1}{i+1} - 2f\binom{n+1}{i} + f\binom{n+1}{i-1}}{\Delta x^2} \tag{A.8}$$

$$f\begin{pmatrix}n\\ i\end{pmatrix} = -Cf\begin{pmatrix}n+1\\ i+1\end{pmatrix} + (1+2C)f\begin{pmatrix}n+1\\ i\end{pmatrix} - Cf\begin{pmatrix}n+1\\ i-1\end{pmatrix} \text{ with } C = D\frac{\Delta t}{\Delta x^2} \tag{A.9}$$

$$\begin{pmatrix} 1+2C & -C & 0 & ... & 0 \\ -C & 1+2C & -C & ... & 0 \\ ... & ... & ... & ... & ... \\ 0 & ... & -C & 1+2C & -C \\ 0 & ... & 0 & -C & 1+2C \end{pmatrix} \begin{pmatrix} f\binom{n+1}{0} \\ f\binom{n+1}{1} \\ ... \\ ... \\ f\binom{n+1}{k} \end{pmatrix} = \begin{pmatrix} f\binom{n}{0} \\ f\binom{n}{1} \\ ... \\ ... \\ f\binom{n}{k} \end{pmatrix} \tag{A.10}$$

Crank Nicolson

$$\frac{\partial T}{\partial t} = D\frac{\partial^2 T}{\partial x^2} \tag{A.11}$$

$$\frac{f\binom{n+1}{i} - f\binom{n}{i}}{\Delta t} = D\frac{f\binom{n}{i+1} - 2f\binom{n}{i} + f\binom{n}{i-1}}{2\Delta x^2} + D\frac{f\binom{n+1}{i+1} - 2f\binom{n+1}{i} + f\binom{n+1}{i-1}}{2\Delta x^2} \tag{A.12}$$

$$-Cf\begin{pmatrix} n+1 \\ i+1 \end{pmatrix} + (1+2C)f\begin{pmatrix} n+1 \\ i \end{pmatrix} - Cf\begin{pmatrix} n+1 \\ i-1 \end{pmatrix} = Cf\begin{pmatrix} n \\ i+1 \end{pmatrix} + (1-2C)f\begin{pmatrix} n \\ i \end{pmatrix} + Cf\begin{pmatrix} n \\ i-1 \end{pmatrix} \tag{A.13}$$

$$\begin{pmatrix} 1+2C & -C & 0 & ... & 0 \\ -C & 1+2C & -C & ... & 0 \\ ... & ... & ... & ... & ... \\ 0 & ... & -C & 1+2C & -C \\ 0 & ... & 0 & -C & 1+2C \end{pmatrix} \begin{pmatrix} f\binom{n+1}{0} \\ f\binom{n+1}{1} \\ ... \\ ... \\ f\binom{n+1}{k} \end{pmatrix} = \tag{A.14}$$

$$\begin{pmatrix} Cf\binom{n}{1} + (1-2C)f\binom{n}{0} + C * Tsurface \\ Cf\binom{n}{2} + (1-2C)f\binom{n}{1} + Cf\binom{n}{0} \\ ... \\ ... \\ C * Tsurface + (1-2C)f\binom{n}{k} + Cf\binom{n}{k-1} \end{pmatrix} \text{ with } C = D\frac{\Delta t}{\Delta x^2} \tag{A.15}$$

Appendix B

Thomas' Algorithm

Thomas' algorithm is an algorithm capable of solving linear system that can be stored in a diagonal matrix. If we take an example:

$$\begin{pmatrix} b & c & ... & & & 0 \\ a & b & c & ... & ... & 0 \\ & & & & & \\ & & & & & \\ 0 & & ... & a & b & c \\ 0 & & & ... & a & b \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ ... \\ ... \\ ... \\ x_k \end{pmatrix} = \begin{pmatrix} d_0 \\ d_1 \\ ... \\ ... \\ ... \\ d_k \end{pmatrix}$$

then we changed the matrix to look like this, by substracting rows, having $c'_i = \frac{c_i}{b_i - a_i c'_{i-1}}$:

$$\begin{pmatrix} 1 & c' & ... & & & 0 \\ 0 & 1 & c' & ... & ... & 0 \\ & & & & & \\ & & & & & \\ 0 & & ... & 0 & 1 & c \\ 0 & & & ... & 0 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ ... \\ ... \\ ... \\ x_k \end{pmatrix} = \begin{pmatrix} d'_0 \\ d'_1 \\ ... \\ ... \\ ... \\ d'_k \end{pmatrix}$$

At this point, we have a one unknown by equation, so the method is just working backward on the equation with $x_i = d'_i x_{i+1}$

Complexity About the complexity, if there is N equations then the matrix will be of size N. It's N operations to transform the matrix and N operations again to go backward. So the complexity is of *O(2N)*