Predicting the number of students in the Netherlands

A. Mooiman

TU Delft

Predicting the number of students in the Netherlands

Master Thesis

By

Arthur Mooiman
Delft University of Technology

Under supervision of:

Prof.dr.ir. C. Vuik
Delft University of Technology

ir. W. van Til
Ministry of Education, Culture and Science

dr.ir L. Wobbes
Ministry of Education, Culture and Science

January 28, 2022

Acknowledgements

Last year has been one of the toughest years in my life. Partly due to the pandemic and partly due to personal circumstances. With help of my supervisors, colleagues and family I managed to keep going and finish my masters degree. I want to give special thanks to my supervisors *Willemijn Schramp-van Til* and *Kees Vuik* for their support and our weekly meetings. Furthermore, I want to give thanks to *Meindert Heres Hoogerkamp* and the rest of my colleagues at the ministry of Education, Culture and Science for our enjoyable lunches and meetings. And lastly, special thanks to my partner for her unwavering support and encouragement.

Arthur Mooiman
Delft, 2022

Abstract

In this thesis the Conjugate Gradient method used in the forecasting of the number of students in the Netherlands is investigated. The addition of international students to the forecasting led to mathematical problem of which a solution could not be computed. While the numerical method is working as intended, the work presented in this thesis shows that the unsolvability lies with a linear system that has no solution. The cause of this is attributed to the use of a selection process in the linear system. Leading to a hugely singular matrix and a forecasting problem without a solution. Several methods were tried to change the use of the selection process but without any results.

Contents

1. Introduction

Education is all about the future. When children enter the educational machine at an age of 4 they can remain there for a time that can easily hit twenty years. To make sure this journey can be travelled without any troubles a plethora of things have to be prepared. The correct books need to be printed, teachers need to be trained and many more tools are necessary. We have to have enough of all of these to facilitate a flawless educational journey. The ministry of Education, Culture and Science (OCW) wants to make sure all these tools are available for any aspiring student. To make sure everything is set the ministry is tasked to look into the future of education. A way in which to do that is to estimate the number of students that are present in the Netherlands for the upcoming years. Based on the estimations the government can set budget to determine a precise amount of resources that need to be available for students.

The prediction of future students has not always been the same as it is today. The ministry of Education, Culture and Science has been predicting the number of students going back as far as 1975. Up until that year the predictions were outsourced ([5], [6]). In those years the fundamentals were laid out for the currently used prediction programs. STUFLO was developed by the Central Planning Bureau ([7]) which was used to predict the level of education of the population. In 1975 a task force, consisting of representatives of Central Planning Bureau, Central Bureau for Statistics and the ministry of Education, Culture and Science, was assigned to create an application to predict how many students are in higher education. The task force created the apps WORSA and RHOBOS ([11]). In the period between 1975 and 1990 there was however a big downside to having multiple applications to predict different parts of education. Most of the applications and results were not compatible with each other and there were large fluctuations in predictions ([6], pp. 6). In between the years of 1990 and 1995 it was decided to no longer outsource the predictions and to calculate a prediction for the whole educational system, not just parts of it. SKILL, previously STUFLO, was integrated and expanded to cover the whole educational apparatus and was renamed to Radon. In the period 1995-2018 not a lot has changed for the prediction process. In 2018 the Referentieramingen team responsible for the predictions expanded significantly after some vulnerabilities of the prediction process were exposed. The team is now working on stabilising and improving Radon.

1.1. Research question

As part of the stabilising and improvement of Radon we will go into greater detail on the solving aspect of the predictions. The solving aspect of Radon has been a black box for quite a while. During the addition of international students to the prediction process there have been several test cases where a solution could

not be found. As a focus of this thesis we will investigate the numerical algorithm that is used and the failing test cases. We will aim to answer the following research question:

1. What causes the numerical methods in the international students test cases to be unable to find a solution?
2. Can the performance of the Conjugate Gradient algorithm be improved within the forecast methodology?

1.2. Thesis outline

In chapter 2 we will expand on the predictions process, what is calculated and what is needed to facilitate those calculations. Next, in chapter 3 we give details on the minimization problem that needs to be solved to get a prediction. A small summary and explanation of the algorithm used to solve the minimization problem is given in chapter 4. The numerical method that is used in the algorithm to calculate the solution and several alternatives are covered in chapter 5. Several examples are showcased in Appendix B for each analysed method. In chapter 6 we dive deeper into the failing test case to figure out why there is no convergence to a solution.

2. Movement in Education

In the Netherlands a large part of the population takes part in some form of education. From children in primary schools and students on universities to evening classes. These are examples of categories in education. And each of these categories has a certain number of people in them. Every year the number of people in each category is counted and recorded in a giant database. At the end of each study year, the students will either stay in their respective education category or move on to another category. The movement of the students that occurs can also be placed in a matrix, where each cell represents the movement from an *origin*[1] category to a *destination* category. This matrix is known as the *education matrix*, a small example is given in Equation 2.1 and a visualisation in Figure 2.1. The education matrix is an integral part in estimating the number of people in each category in the upcoming years. If we can calculate the education matrix, then we know how many students are present in the next year. Sadly there have been some cases where an optimal education matrix cannot be calculated.

$$\begin{array}{cc} & \textbf{Destination} \\ \textbf{Origin} & \begin{pmatrix} 1 & 2 \\ 0 & 2 \end{pmatrix} \end{array} \tag{2.1}$$

An example of an education matrix. The movement from origin 1 to destination 1 is equal to 1. From origin 1 to destination 2 is equal to 2. from origin 2 to destination 2 is equal to 2. There is no movement from origin 2 to destination 1.

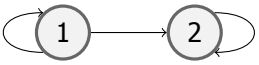


Figure 2.1 *A schematic visualisation of the education matrix.*

2.1. Radon

The Referentieramingen team of The ministry of Education, Culture and Science is the team which is designated to predict the number of students in each category for at least several years into the future. For this they use a program, Radon, which calculates the education matrix. We will go into greater detail about the calculation in chapter 3. For the calculation Radon requires the data of previous years. Among these are the education matrices and the origin and the destination vectors of those years. During the calculation Radon extrapolates a rough estimation of the next years education matrices, which are based on graduation chances and older education matrices. Based on this estimation we can solve the minimization prob-

[1] Further definitions of words that are printed in italic can be found in Appendix A

lem. The goal of the algorithm is to calculate the education matrix for upcoming years.

2.1.1. Constraints

There are a multitude of constraints which are imposed on the problem. One of the constraints will specify that each person in an origin category will also end up in a destination category. If this is not enforced then it would be possible for students to vanish from the system as they could leave a origin category and not end up in a destination category. Another constraint is that each person who earns a diploma will end up in a non-diploma destination category. These constraints are necessary for the problem to be solved. Furthermore there are two types of constraints that will influence the education matrix. As they either force a number of students to end up in one or more destination categories or they force a percentage of students from an origin category to move to one or more destination categories. These constraints are called absolute and relative constraints respectively. An *absolute constraint* is a predetermined control value on how many people can end up in one or more destination categories. For instance, we can demand of Radon that a total of 200 000 people will end up in the MBO (Dutch college). *Relative constraints* are also predetermined control values but instead of giving a hard number, we give a percentage. An example of this is if we want that 50 % of graduating VWO (pre-university education) students will go to University. Then half of the students in the category graduating VWO is forced to University.

2.1.2. The minimization problem

Now that we have seen what information and constraints are used we can take a look at a summarized version of the forecasting problem. Which is as follows:

$$\begin{aligned} &\min_{\mathbf{x}} \|\mathbf{x} - \mathbf{a}\|_W \\ &\textbf{subject to} \\ &A\mathbf{x} = \mathbf{b} \\ &S^*\mathbf{x} \geq 0 \\ &0 \leq W \leq \infty \end{aligned} \tag{2.2}$$

Where $A \in \mathbb{R}^{n \times m}$ is the matrix for the constrains, consisting of handshaking and diploma equations and control values. $S^* \in \mathbb{R}^{m \times m}$ the indicator matrix to select certain categories and control values. Initially this selector selects every variable in the education matrix $\mathbf{x}$. $\mathbf{a} \in \mathbb{R}^n$ is the target education matrix. $\mathbf{b} \in \mathbb{R}^n$ the vector with origin values and control values. And $\mathbf{x} \in \mathbb{R}^n$ the to be calculated education matrix to minimize the problem.

This minimization problem will minimize the difference between the extrapolated education matrix $\mathbf{a}$ and the to be calculated education matrix $\mathbf{x}$. Leading to an education matrix that is close to the extrapolated version while also considering the

constraints imposed on the problem. In the following sections we will refer to this problem as the forecasting problem.

3. The forecasting problem

A summarized version of the minimization problem has been shown in Equation 2.2. The goal of the forecasting problem is to minimize the differences between the solution education matrix in vector form $\mathbf{x}$ and the extrapolated education matrix in vector form $\mathbf{a}$ while also making sure the controlled variables are close to their control value. $\mathbf{a}$ and $\mathbf{x}$ are in fact vectors while we refer to them as what they represent, the education matrix. We note the full minimization problem that has to be solved to calculate an optimal education matrix $\mathbf{x}$:

$$\min_{(\mathbf{x},\hat{Y})} \sum_{ijk} g_{ijk} \left(x_{ijk} - a_{ijk}\right)^2 + \sum_{q} \mathbf{G}_q \left(\hat{\mathbf{Y}}_q - \mathbf{D}_q\right)^2 \tag{3.1}$$

Under the following conditions:

$$\sum_{j} x_{ijk} = H_{ik} \text{ For each non-graduation category } i \text{ and each } k \tag{3.2}$$

$$\sum_{i} x_{idk} = \sum_{j} x_{djk} \text{ For each graduation category } d \text{ and each } k \tag{3.3}$$

In Equation 3.1 there are four different indices. i and j are the indices for origin category and destination category. Index k is used to categorise people based on their age. Then an element of the education matrix x_{ijk} is the movement from origin i to destination j for a given age group k. The last index q is just a numbering for the amount of control equations imposed on the minimization problem. Since a different education matrix is calculated for each age, we can set age to a constant and drop it from notation.

In the minimization problem we write the education matrix as a vector. An example of this with a 3×3 education matrix is given in Equation 3.4 and Equation 3.5:

$$Y = \begin{pmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ y_{31} & y_{32} & y_{33} \end{pmatrix} \tag{3.4}$$

And its vector form:

$$\hat{\mathbf{Y}} = \text{vec}(Y) = \begin{pmatrix} y_{11} \\ y_{12} \\ y_{13} \\ y_{21} \\ y_{22} \\ y_{23} \\ y_{31} \\ y_{32} \\ y_{33} \end{pmatrix} \tag{3.5}$$

A list and explanation of all the variables that are present in the forecasting Equation 3.1 can be found in Table 3.1. In the following sections we will cover some of these variables.

Table 3.1 *Variables of the forecasting problem.*

$x_{ij} \in \mathbb{R}^{m \times n}$	Elements of the to be calculated education matrix.
$a_{ij} \in \mathbb{R}^{m \times n}$	Elements of the extrapolated education matrix.
$g_{ij} \in \mathbb{R}^{m \times n}$	Weights for the education matrix.
$\widehat{\mathbf{Y}}_q \in \mathbb{R}^k$	Control values. Where $q \in \{1, 2, \dots, k\}$
$\mathbf{D}_q \in \mathbb{R}^k$	The predetermined control values. Where $q \in \{1, 2, \dots, k\}$
$\mathbf{G}_q \in \mathbb{R}^k$	Weights for the control values. Where $q \in \{1, 2, \dots, k\}$
$H_{ij} \in \mathbb{R}^{m \times n}$	Origin category vector values.

3.1. Estimate the education matrix

In the minimization problem we need an estimate education matrix $\mathbf{a}$ of the to be determined education matrix $\mathbf{x}$. This estimation is done by multiplying the origin values H_{ij} by the graduation chance p_{ij},

$$a_{ij} = p_{ij} H_{ij} \tag{3.6}$$

The graduation chance of the to be calculated year will be extrapolated from the previous years. The extrapolation will mostly only be used for *strategic streams* in education [1]. For example from VWO graduation to university. The extrapolation that is used in Radon is Pseudo Ordinary Least Squares (POLS). The graduation chances also need to be weighted. The weight of a graduation chance is based on the goodness of fit of the trend of the extrapolated values.

3.1.1. Extrapolating with POLS

In this section we will cover the Pseudo Ordinary Least Squares (POLS) method for extrapolating graduation chances p_{ij} which is used by Radon. POLS acts like Ordinary Least Squares (OLS) but translates the resulting linear function such that it crosses the last known graduation chance. The extrapolated graduation chance is needed to estimate $\mathbf{a}$. In Figure 3.1 we have an example of an extrapolation made by POLS. The green dotted line is the POLS extrapolation that is used to determine the graduation chances.

OLS is a linear least squares method for finding the unknown parameters in a linear regression model. The method minimizes the sum of the squared residuals, the vertical distance from an observation to the predicted linear function, to estimate the parameter which is used in linear regression. The linear regression model takes

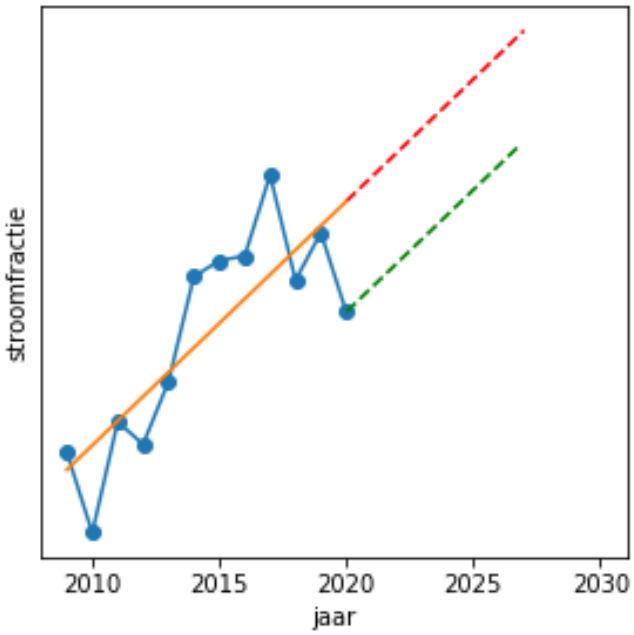


Figure 3.1 *An example of POLS. The orange line is a linear interpolation calculated by OLS which is then extrapolated on the red dotted line. In POLS we place the extrapolation along the last known value. Resulting in the green dotted line.*

the form:

$$\mathbf{y} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon} \tag{3.7}$$

Where $\mathbf{y} \in \mathbb{R}^n$ is the vector of observed values, in this case the previous graduation chances; $X \in \mathbb{R}^{n \times p}$ the design matrix consisting of $\mathbf{x}_i \in \mathbb{R}^n$ column vectors of regressors; $\boldsymbol{\beta} \in \mathbb{R}^p$ a parameter vector and $\boldsymbol{\varepsilon} \in \mathbb{R}^n$ the error term. The goal is to find $\boldsymbol{\beta}$ such that the following equation is minimized:

$$\hat{\boldsymbol{\beta}} = S(\boldsymbol{\beta}) = \|\mathbf{y} - X\boldsymbol{\beta}\|^2 \tag{3.8}$$

Radon estimates β with:

$$\boldsymbol{\beta} = \frac{\sum_t (t - \hat{t})(s_t - \hat{s})}{\sum_t (t - \hat{t})^2} \tag{3.9}$$

Where $\hat{t}$ is the average over the graduation chances of the observation years, and $\hat{s}$ the average over the series of graduation chances. The extrapolated graduation chances are also weighted according to their goodness of fit. Having a higher weight for a greater goodness of fit. The data of older observation years also get a lower weight attached to them.

3.2. Adding the constraints

We also need to add the constraints that were mentioned in subsection 2.1.1 into the minimization problem. this is done by adding Equation 3.10 to the minimization of the education matrix.

$$\sum_{q} \mathbf{G}_q \left(\widehat{\mathbf{Y}}_q - \mathbf{D}_q\right)^2 \tag{3.10}$$

In this case we have that

$$\widehat{\mathbf{Y}}_q = \sum_{r \in \mathbf{S}_q} x_r$$

is the sum of the to be controlled variables. We determine that this sum has to have the value of D_q. Where S_q is the set of variables in the education matrix that is controlled.

3.3. Applying weights

Weight vectors $\mathbf{g} \in \mathbb{R}^n$ and $\mathbf{G} \in \mathbb{R}^k$ are the vectors containing weights for the elements of the education matrices and the control values. We can use the weights to influence how much a flow or constraint in education should be realised. A higher weight will make sure an absolute or relative constraint is enforced to full effect. An infinite weight on a constraint is the highest form there is, but is hardly ever used. A lower weight will be more lenient on the enforcement. The weights are applied in the minimization problem for both the education matrices and the constraints.

3.4. Applying additional selection

In addition to the initial selection, as explained in subsection 2.1.2, another selection may be used during several parts of the calculation. This selection is made on elements of the education matrix that have to be optimized further, in case they do not fulfill certain constraints of the problem. The selection matrix $S \in \mathbb{R}^{n+k}$ consists only of zeroes and ones. Where there is a 1 for the category that needs to be included in the calculation. At the start of the calculation everything is taken into account so S will be the identity vector.

4. Calculating an optimal education matrix

To calculate an optimal education matrix multiple steps need to be taken. Not all of them are solving linear systems with numerical methods. In this chapter we will go over the steps that are taken and the different parts of the algorithm that make it happen.

4.1. The steps of the algorithm

The problem that needs to be solved is of course the Forecasting Problem. For simplicity we note it again in Equation 4.1.

$$\begin{aligned} &\min_{\mathbf{x}} \|\mathbf{x} - \mathbf{a}\|_W \\ &\textbf{subject to} \\ &A\mathbf{x} = \mathbf{b} \\ &S^*\mathbf{x} \geq 0 \\ &0 \leq W \leq \infty \end{aligned} \tag{4.1}$$

The solution to the forecasting problem is an optimal education matrix $\mathbf{x}$ for an upcoming year. The first step to take is to transform the problem (given in Equation 4.1) into another problem that is easier to solve. By substituting $\mathbf{y} = \mathbf{x} - \mathbf{a}$ the problem transforms to minimizing $\mathbf{y}$. This problem still minimizes the difference between $\mathbf{x}$ and $\mathbf{a}$ but allows us to move known variables around in the constraints. We end up with the following problem:

$$\begin{aligned} &\min_{\mathbf{y}} \|\mathbf{y}\|_W \\ &\textbf{subject to} \\ &A\mathbf{y} = \mathbf{b} - A\mathbf{a} \\ &S^*(\mathbf{y} + \mathbf{a}) \geq 0 \\ &0 \leq W \leq \infty \end{aligned} \tag{4.2}$$

The transformed problem is not directly solved with all the constraints and selection in mind. First an initial solution $\mathbf{y}_*$ is calculated as a rough starting point. In this initial solution we do not force any of the constraints or selection. $\mathbf{y}_*$ is the solution to the following problem

$$\begin{aligned} &\min_{\mathbf{y}} \|\mathbf{y}\|_W \\ &\textbf{subject to} \\ &A\mathbf{y} = \mathbf{b} - A\mathbf{a} \\ &0 \leq W \leq \infty \end{aligned} \tag{4.3}$$

The initial solution vector $\mathbf{y}_*$ is a lot easier to calculate as we do not have to take into account the additional constraints. Radon will improve the initial solution and

calculate $\mathbf{z}$. $\mathbf{z}$ is calculated by adding back the constraints to the minimization problem and starting the numerical calculation with starting vector $\mathbf{y}_*$. This optimization comes down to transforming the forecasting problem even further with $\mathbf{z} = \mathbf{y} - \mathbf{y}_*$ to:

$$\begin{gathered} \min_{\mathbf{z}} \|\mathbf{z}\|_W \\ \textbf{subject to} \\ A\mathbf{z} = \mathbf{b} - A\mathbf{a} - A\mathbf{y}_* \\ S^*(\mathbf{z} + \mathbf{y}_* + \mathbf{a}) \geq 0 \\ 0 \leq W \leq \infty \end{gathered} \tag{4.4}$$

The resulting solution $\mathbf{z}_*$ is optimal. The solution to minimization problem Equation 4.1 $\mathbf{x}_*$ can be easily calculated with $\mathbf{x}_* = \mathbf{z}_* + \mathbf{a} + \mathbf{y}_*$.

4.2. The algorithms of Radon

We will have a closer look at the code of Radon and its algorithms, as explained in the technological manual [1]. The algorithm that is called upon to solve the minimization problem (given in Equation 4.1) is called GBLDP. A small summary and abbreviation explanation can be found in Figure 4.1 and Table 4.1. Let us first list all the algorithms that are used by Radon.

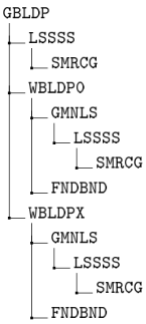


Figure 4.1 *A summary of the Radon algorithm.*

GBLDP	Generalized Bounded Linear Distance Programming
LSSSS	Least Squares Solution Singular Symmetric System
SMRCG	Scaled Minimum Residual Conjugate Gradients
WBLDP0	Weighted Bounded Linear Distance Programming (Start at 0)
WBLDPX	Weighted Bounded Linear Distance Programming
GMNLS	Generalized Minimum Norm Least Squares
FNDBND	Find Boundary

Table 4.1 *Explanation of the abbreviations used in Figure 4.1*

4.2.1. GBLDP

GBLDP (Generalized Bounded Linear Distance Programming, [1], p. 23) is the main algorithm that solves the minimization problem (given in Equation 4.1) by calling the other algorithms along the way. It follows the steps described in the previous sections. GBLDP first calls LSSSS (subsection 4.2.2) to solve the minimization problem (given in Equation 4.3), calculating $\mathbf{y}_*$.

4.2.2. LSSSS

LSSSS (Least Squares Solution Singular Symmetric System, [1], p. 24) sets up for the conjugate gradient algorithm (SMRCG, 4.2.3) and does some post-processing. It receives the minimization problems (given in Equation 4.3 and Equation 4.4) and the optimization of $\mathbf{z}_*$.

4.2.3. SMRCG

SMRCG (Scaled Minimum Residual Conjugate Gradients, [1], p. 27) is the conjugate gradient algorithm used to calculate a solution to the linear system

$$A\mathbf{x} = \mathbf{b}. \tag{4.5}$$

The conjugate gradient method is the main focus of this thesis. As the Conjugate Gradient method takes care of the calculation it is only logical to examine this first. We will investigate how conjugate gradient works in chapter 5. SMRCG can be called multiple times by LSSSS, each time $\mathbf{b}$ can be altered slightly, getting closer to the optimal solution to the given minimization problem. The calculated solutions are returned to GBLDP for further processing.

4.2.4. WBLDP0 and WBLDPX

WBLDP0 (Weighted Bounded Linear Distance Programming, [1], p. 29) is called by GBLDP to initialize the calculation of the minimization problem given in Equation 4.4. The calculation starts with an initial solution vector $\mathbf{x}_0 = \mathbf{0}$. WBLDP0 stops once a feasible solution $\mathbf{z}_*$ has been found. The algorithm will transform the minimization problem (given in Equation 4.3) to the minimization problem given in Equation 4.4. First it will check if $\mathbf{y}_*$ satisfies all the added selection and weight constraints. If that is the case then Radon can move on to the next algorithm WBLDPX. Any value in $\mathbf{y}_*$ that doesn't satisfy the constraints is replaced by the lower bound for that constraint. At this point GMNLS (subsection 4.2.5) is called to minimize $\|\mathbf{z}\|_W$ and in turn the minimization problem of Equation 4.4. After WBLPD0 has finished we end up with a feasible solution $\mathbf{z}_*$. WBLDPX continues where WBLDP0 stops and will minimize $\|\mathbf{z} - \mathbf{z}_*\|_W$. Giving an optimal solution for $\mathbf{z}_*$ and for $\mathbf{x}_* = \mathbf{z}_* + \mathbf{a} + \mathbf{y}_*$.

4.2.5. GMNLS

GMNLS (Generalized Minimum Norm Least Squares, [1], p. 31) determines the solution vector $\mathbf{z}_*$ of which $\|\mathbf{z}\|_W$ is minimized for the problem given in Equation 4.4. GMNLS algorithm also takes into account that any changed variables in $\mathbf{y}_*$ by WBLDP0 satisfies the constraints. The calculation is again done via LSSSS and SMRCG, solving the minimization problem given in Equation 4.4. Additionally GMNLS is used to optimize solution $\mathbf{z}_*$.

4.2.6. FNDBND

The FNDBND (Find Boundary, [1], p. 31) algorithm is executed after a solution has been found that satisfies all the constraints. It will iterate over each value in the solution and check if that value is on the boundary of the solution field. If that is not the case then that value is moved to the closest boundary. This changes any satisfied constraint that is an inequality to an equality, minimizing $\|\mathbf{z}_*\|$ even further.

5. Numerically solving the forecasting problem

In this section we will be taking a look at how Radon solves the linear system

$$A\mathbf{x} = \mathbf{b}, \tag{5.1}$$

where $A \in \mathbb{R}^{m \times n}$, $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}^m$ with $m \ll n$. This system is known as an underdetermined system, meaning there are more variables than equations. To calculate a solution to Equation 5.1 Radon uses the method Conjugate Gradient. We will investigate the Conjugate Gradient method in section 5.1 and research additional methods to solve Equation 5.1, CGNE in subsection 5.1.4 and LSQR in section 5.3.

5.1. Conjugate Gradient

Conjugate Gradient works if matrix A is square, symmetrical and positive semi-definite. In general, the matrix A for which Radon has to solve the linear system is not square or symmetrical. So instead of solving $A\mathbf{x} = \mathbf{b}$ Radon will solve the linear system $AA^T\mathbf{u} = \mathbf{b}$ where $\mathbf{x} = A^T\mathbf{u}$. For the calculation Radon uses the standard Conjugate Gradient method, explained in subsection 5.1.2. We discuss two alternative methods to solve an underdetermined system. The first is CGNE, a method based on Conjugate Gradient, which is explained in subsection 5.1.4. The other method is LSQR, explained in section 5.3. We will first cover what type of method Conjugate Gradient is in the section below.

5.1.1. Krylov subspace

The Conjugate Gradient method is a Krylov-subspace iterative method. The Krylov-subspace is defined as follows (from [13]):

Definition 5.1.1 (Krylov-subspace) *Let $A \in \mathbb{R}^{n \times n}$ and $\mathbf{r}_0 \in \mathbb{R}^n$. Then the space $K^k(A; \mathbf{r}_0)$ as defined by:*

$$K^k(A; \mathbf{r}_0) := span\{\mathbf{r}_0, A\mathbf{r}_0, ..., A^{k-1}\mathbf{r}_0\}$$

Is a Krylov-subspace of dimension k corresponding to matrix A and initial residue $\mathbf{r}_0$.

This definition follows from the basic iterative methods for the linear system $A\mathbf{x} = \mathbf{b}$ with the following recursion:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + (\mathbf{b} - A\mathbf{x}_k) = \mathbf{x}_k + \mathbf{r}_k$$

Which implies that

$$\mathbf{x}_k \in \mathbf{x}_0 + \text{span}\{\mathbf{r}_0, A\mathbf{r}_0, ..., A^{k-1}\mathbf{r}_0\}.$$

Meaning that we can find our solution in the Krylov-subspace.

5.1.2. Standard Conjugate Gradient

In this section we will describe the Conjugate Gradient method in greater detail. To facilitate the formulas of Conjugate Gradient we will assume $\mathbf{x}_0 = 0$ so $\mathbf{r}_0 = \mathbf{b}$. These assumptions are not necessary for the Conjugate Gradient method itself but are for ease of notation. We will furthermore assume that matrix A is symmetric and positive definite. Which are both necessary conditions for Conjugate Gradient and defined as follows.

Definition 5.1.2 *symmetric*
The square matrix $A \in \mathbb{R}^{n\times n}$ is called symmetric if and only if $A = A^T$

Definition 5.1.3 *Positive definite*
The square matrix $A \in \mathbb{R}^{n\times n}$ is called positive definite if and only if

$$\forall \mathbf{x} \in \mathbb{R}^n \setminus \{0\} : \mathbf{x}^T A\mathbf{x} > 0 \tag{5.2}$$

Conjugate Gradient is an iterative method which finds a solution to the following minimization problem:

$$||\mathbf{x} - \mathbf{x}_k||_A = \min_{\mathbf{y}\in K^k(A;\mathbf{r}_0)} ||\mathbf{x} - \mathbf{y}|| \tag{5.3}$$

Where we use the following definition for the used A-inner product and A-norm:

Definition 5.1.4 *A-inner product*
The A-inner product is defined by

$$(\mathbf{x}, \mathbf{u})_A = \mathbf{x}^T A\mathbf{u} \tag{5.4}$$

Definition 5.1.5 *A-norm*
Let $A \in \mathbb{R}^{n\times n}$, then A-norm of a vector $\mathbf{x} \in \mathbb{R}^n$ is defined by:

$$||\mathbf{x}||_A = \sqrt{(\mathbf{x}, \mathbf{x})_A} = \sqrt{\mathbf{x}^T A\mathbf{x}} \tag{5.5}$$

When we want to minimize $||\mathbf{x} - \mathbf{x}_1||_A^2$ in the A-norm we have

$$||\mathbf{x} - \mathbf{x}_1||_A^2 = \mathbf{x}^T A\mathbf{x} - 2\alpha_0(\mathbf{r}_0)^T A\mathbf{x} + \alpha_0^2(\mathbf{r}_0)^T A\mathbf{r}_0, \tag{5.6}$$

Where α_0 is a constant that has to be chosen such that $||\mathbf{x}-\mathbf{x}_1||_A^2$ is minimal. Which happens when $\alpha_0 = \frac{(\mathbf{r}_0)^T\mathbf{b}}{(\mathbf{r}_0)^T A\mathbf{r}_0}$. These inner products can be solved as $\mathbf{r}_0, \mathbf{b}$ and A are known, leading to the Conjugate Gradient method for solving Equation 5.3. The steps for the standard Conjugate Gradient method are given below.

Input : $A \in \mathbb{R}^{n\times n}, \mathbf{x}_0 \in \mathbb{R}^n, \mathbf{b} \in \mathbb{R}^n$
$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$;
$\mathbf{p}_0 = \mathbf{r}_0$
$i = 0$
while $i < i_{max}$ ***AND*** $\frac{\|\mathbf{r}_i\|}{\|\mathbf{b}\|} > \varepsilon$ **do**
  $\alpha_i := \frac{(\mathbf{r}_i,\mathbf{r}_i)}{(A\mathbf{p}_i,\mathbf{p}_i)}$;
  $\mathbf{x}_{i+1} := \mathbf{x}_i + \alpha_i\mathbf{p}_i$;
  $\mathbf{r}_{i+1} := \mathbf{r}_i - \alpha_i A\mathbf{p}_i$;
  $\beta_i := \frac{(\mathbf{r}_{i+1},\mathbf{r}_{i+1})}{(\mathbf{r}_i,\mathbf{r}_i)}$;
  $\mathbf{p}_{i+1} := \mathbf{r}_{i+1} + \beta_i\mathbf{p}_i$;
end

Algorithm 1: Conjugate Gradient

Since the forecasting problem has a few instances where the Conjugate Gradient algorithm does not converge, we will take a look at the rate of convergence of the method. These characteristics may give is valuable information on why there is no convergence. From [13] we have Theorem 5.1.1 on an upper bound of the difference between the i^{th}-solution $\mathbf{x}_i$ and the exact solution $\mathbf{x}$. Where we make use of the following notation:

Definition 5.1.6 *Spectrum of a matrix*
The set of eigenvalues of a matrix A is called the spectrum of A and is denoted by

$$\sigma(A) = \{\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n\} \tag{5.7}$$

Definition 5.1.7 *Max and Min eigenvalues*
The maximum eigenvalue of matrix A is defined as

$$\lambda_{\max}(A) = \max_{i=1,\ldots,n} \{|\lambda_i| : \lambda_i \in \sigma(A)\} \tag{5.8}$$

The minimum eigenvalue of matrix A is defined as

$$\lambda_{\min}(A) = \min_{i=1,\ldots,n} \{|\lambda_i| : \lambda_i \in \sigma(A)\} \tag{5.9}$$

Definition 5.1.8 *Condition number*
The condition number of A in the L^p-norm is noted by

$$\kappa_p(A) = \|A\|_p \|A^{-1}\|_p. \tag{5.10}$$

The condition number of A in the L^2-norm is noted by

$$\kappa_2(A) = \frac{\sqrt{\lambda_{max}(A^T A)}}{\sqrt{\lambda_{min}(A^T A)}} \tag{5.11}$$

Theorem 5.1.1 *The iterates $\mathbf{x}_i$ obtained from the Conjugate Gradient algorithm satisfy the following inequality:*

$$||\mathbf{x} - \mathbf{x}_i||_A \leq 2\left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1}\right)^i ||\mathbf{x} - \mathbf{x}_0||_A \tag{5.12}$$

The condition number of a matrix can tell us a lot about the convergence of Conjugate Gradient or any other numerical method. As we can see in Equation 5.11, a condition number $\kappa_2(A) = 1$ is achieved when $\lambda_{\text{max}} = \lambda_{\text{min}}$. That is when all the eigenvalues of A are equal. When this happens we can conclude from Theorem 5.1.1 that $||\mathbf{x} - \mathbf{x}_i||_A = 0$, giving instant convergence. So it is desirable for a matrix A to have its eigenvalues close to each other. If it is the case that a matrix A has $\lambda_{\text{min}} = 0$ then $\kappa_2(A) = \infty$ giving no convergence. $\lambda_{\text{min}} = 0$ occurs when A is singular.

5.1.3. Normal Equation Conjugate Gradient

Radon solves a different form of linear system than in subsection 5.1.2. It sets $\mathbf{x} = A^T\mathbf{u}$, so it solves for $\mathbf{u}$

$$AA^T\mathbf{u} = \mathbf{b}. \tag{5.13}$$

The linear system in Equation 5.13 is used to solve under-determined systems. Which is what we have in the forecasting problem. Where $A \in \mathbb{R}^{m \times n}$ is a rectangular matrix with $m \ll n$. This system is known as the system of *normal equations* which minimizes the least-squares problem

$$\min||\mathbf{x}_* - A^T\mathbf{u}||_2. \tag{5.14}$$

Where $\mathbf{x}_*$ is any valid solution to $A\mathbf{x} = \mathbf{b}$. Since we defined $A^T\mathbf{u} = \mathbf{x}$ we will find an $\mathbf{x}$ that is closest to $\mathbf{x}_*$ in the 2-norm. Conjugate Gradient methods that solve these kind of problems are known as CGNE. Where 'N' stands for the normal equation and 'E' for 'Error', solving the system by minimising the error.

In the following section we will describe the general details of CGNE. A more in depth description can be found in [10].

5.1.4. CGNE

CGNE is used to solve the linear system in Equation 5.13, where matrix A is underdetermined. We will once again minimize $\|\mathbf{x}-\mathbf{x}_i\|$ but now in the AA^T-norm. Showing that this is the same as minimizing the error in the 2-norm:

$$\begin{aligned} ||\mathbf{x}-\mathbf{x}_i||^2_{AA^T} &= (A^T\mathbf{u}-A^T\mathbf{u_i})^T AA^T(A^T\mathbf{u}-A^T\mathbf{u_i}) \\ &= (A(\mathbf{u}-\mathbf{u}_i))^T(A(\mathbf{u}-\mathbf{u}_i)) \\ &= (\mathbf{x}-\mathbf{x}_i)^T(\mathbf{x}-\mathbf{x}_i) \\ &= ||\mathbf{e}_i||_2^2. \end{aligned} \tag{5.15}$$

Finding a solution in the subspace $\mathbf{x}_0 + K_m(AA^T, \mathbf{r}_0)$.

To construct the algorithm we start by applying the Conjugate Gradient method directly to the linear system in Equation 5.13 with $\mathbf{q}_i$, the conjugate direction, to get the following Conjugate Gradient iteration:

$$\begin{aligned} \alpha_i &:= \frac{(\mathbf{r}_i,\mathbf{r}_i)}{(AA^T\mathbf{q}_i,\mathbf{q}_i)} \\ \mathbf{x}_{i+1} &:= \mathbf{x}_i + \alpha_i\mathbf{q}_i \\ \mathbf{r}_{i+1} &:= \mathbf{r}_i - \alpha_i A\mathbf{q}_i \\ \beta_i &:= \frac{(\mathbf{r}_{i+1},\mathbf{r}_{i+1})}{(\mathbf{r}_i,\mathbf{r}_i)} \\ \mathbf{p}_{i+1} &:= \mathbf{r}_{i+1} + \beta_i\mathbf{q}_i \end{aligned} \tag{5.16}$$

We can rewrite these iterations in the original vector $\mathbf{x}_i = \mathbf{x}_0 + A^T(\mathbf{u}_i - \mathbf{u}_0)$ by introducing $\mathbf{p}_i = A^T\mathbf{q}_i$. With this substitution the residual vectors for $\mathbf{x}$ and $\mathbf{u}$ are the same and can be obtained with $\mathbf{p}_{i+1} := A^T\mathbf{r}_{i+1} + \beta_i\mathbf{p}_i$. The resulting algorithm (CGNE) is also known as Craig's method.

Input : $A \in \mathbb{R}^{m\times n}, \mathbf{x}_0 \in \mathbb{R}^n, \mathbf{b} \in \mathbb{R}^m$
$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$;
$\mathbf{p}_0 = A^T\mathbf{r}_0$
$i = 0$
while $i < i_{max}$ ***AND*** $\frac{\|\mathbf{r}_i\|}{\|\mathbf{b}\|} > \varepsilon$ **do**
  $\alpha_i := \frac{(\mathbf{r}_i,\mathbf{r}_i)}{(\mathbf{p}_i,\mathbf{p}_i)}$;
  $\mathbf{x}_{i+1} := \mathbf{x}_i + \alpha_i\mathbf{p}_i$;
  $\mathbf{r}_{i+1} := \mathbf{r}_i - \alpha_i A\mathbf{p}_i$;
  $\beta_i := \frac{(\mathbf{r}_{i+1},\mathbf{r}_{i+1})}{(\mathbf{r}_i,\mathbf{r}_i)}$;
  $\mathbf{p}_{i+1} := A^T\mathbf{r}_{i+1} + \beta_i\mathbf{p}_i$;
end

Algorithm 2: CGNE (Craig's method)

5.2. Convergence of the Conjugate Gradient method

Iterative methods like Conjugate Gradient and LSQR are often lacking in speed of convergence. One way to improve the convergence speed and robustness is to use preconditioning. The preconditioner is a matrix that transforms the original linear system such that the transformed system is easier to converge to a solution. We will first cover the preconditioner of Conjugate Gradient in subsection 5.2.1 and after that we will cover the preconditioner of CGNE in subsection 5.2.2.

5.2.1. Conjugate Gradient preconditioner

Radon uses a simple form of preconditioning, diagonal preconditioning, for Conjugate Gradient which is described in Golub and van Loan [4]. Instead of solving $M\mathbf{x} = \mathbf{b}$ it will solve the following:

$$N\mathbf{y} = \mathbf{f} \tag{5.17}$$

where

$$\begin{aligned} N &= D^{-1}MD^{-1}, \\ \mathbf{y} &= D\mathbf{x}, \\ \mathbf{f} &= D^{-1}\mathbf{b} \end{aligned}$$

In this preconditioning D is a diagonal matrix such that diag(N)=$\mathbf{I}$, where $\mathbf{I}$ is the identity vector. This allows us to calculate the diagonal elements of D with

$$D_{ii} = \sqrt{M_{ii}}.$$

Giving an easy calculation of D and its inverse D^{-1}.

Input : $M \in \mathbb{R}^{m \times n}, \mathbf{x}_0 \in \mathbb{R}^n, \mathbf{b} \in \mathbb{R}^n$
$\mathbf{r}_0 = \mathbf{b} - M\mathbf{x}_0$;
$\mathbf{z}_0 = D^{-1}\mathbf{r}_0$
$\mathbf{p}_0 = \mathbf{z}_0$
$i = 0$
while $i < i_{max}$ ***AND*** $\frac{\|\mathbf{r}_i\|}{\|\mathbf{b}\|} > \varepsilon$ **do**
  $\alpha_i := \frac{(\mathbf{r}_i, \mathbf{z}_i)}{(M\mathbf{p}_i, \mathbf{p}_i)}$;
  $\mathbf{x}_{i+1} := \mathbf{x}_i + \alpha_i \mathbf{p}_i$;
  $\mathbf{r}_{i+1} := \mathbf{r}_i - \alpha_i M\mathbf{p}_i$;
  $\mathbf{z}_{i+1} := D^{-1}\mathbf{r}_{i+1}$
  $\beta_i := \frac{(\mathbf{r}_{i+1}, \mathbf{z}_{i+1})}{(\mathbf{r}_i, \mathbf{z}_i)}$;
  $\mathbf{p}_{i+1} := \mathbf{z}_{i+1} + \beta_i \mathbf{p}_i$;
end

Algorithm 3: Preconditioned Conjugate Gradient

5.2.2. CGNE preconditioner

The preconditioner that is used for Conjugate Gradient, $D_{ii} = \sqrt{M_{ii}}$, can not be used for non-square matrices. So for CGNE, which we use for the rectangular matrices, we need a different preconditioner. Instead of the regular system $A\mathbf{x} = \mathbf{b}$ we will solve

$$N\mathbf{x} = \mathbf{f} \tag{5.18}$$

where

$$N = D^{-1}A\mathbf{f} = D^{-1}\mathbf{b} \tag{5.19}$$

In this preconditioning D is the diagonal matrix such that for $A \in \mathbb{R}^{m\times n}$ we can calculate $D \in \mathbb{R}^{m\times m}$ with

$$D_{ii} = \sum_{j=1}^{n} |a_{ij}| \tag{5.20}$$

We can use the preconditioning matrix in the following algorithm.

Input : $A \in \mathbb{R}^{m\times n}, \mathbf{x}_0 \in \mathbb{R}^n, \mathbf{b} \in \mathbb{R}^n$
$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$;
$\mathbf{z}_0 = D^{-1}\mathbf{r}_0$
$\mathbf{p}_0 = A^T\mathbf{z}_0$
$i = 0$
while $i < i_{max}$ ***AND*** $\frac{\|\mathbf{r}_i\|}{\|\mathbf{b}\|} > \varepsilon$ **do**
  $\mathbf{w}_i = A\mathbf{p}_i$
  $\alpha_i := \frac{(\mathbf{r}_i,\mathbf{z}_i)}{(\mathbf{p}_i,\mathbf{p}_i)}$;
  $\mathbf{x}_{i+1} := \mathbf{x}_i + \alpha_i\mathbf{p}_i$;
  $\mathbf{r}_{i+1} := \mathbf{r}_i - \alpha_i\mathbf{w}_i$;
  $\mathbf{z}_{i+1} := D^{-1}\mathbf{r}_{i+1}$
  $\beta_i := \frac{(\mathbf{r}_{i+1},\mathbf{z}_{i+1})}{(\mathbf{r}_i,\mathbf{z}_i)}$;
  $\mathbf{p}_{i+1} := A^T\mathbf{z}_{i+1} + \beta_i\mathbf{p}_i$;
  $i = i + 1$
end

Algorithm 4: Preconditioned CGNE

5.3. LSQR

Another numerical method for solving the system $A\mathbf{x} = \mathbf{b}$ is LSQR [9]. LSQR requires that $A \in \mathbb{R}^{m\times n}$ and $\mathbf{b} \in \mathbb{R}^m$ are real. Similar to Conjugate Gradient, LSQR is an iterative method. Reaching an acceptable approximation of $\mathbf{x}$ in around n iterations, but that may vary based on the problem. A faster convergence is achieved if matrix A is well conditioned with eigenvalues close to each other. A

preconditioner can be used to transform the system to get the eigenvalues closer together.

LSQR is based on the bidiagonalization procedure of Golub and Kahan [3] and the symmetric Lanczos process [8]. In each iteration the algorithm calculates $\mathbf{x}_i$ such that $\|\mathbf{r}\|_2$ decreases monotonically. The Lanczos process generates a sequence of vector $\mathbf{v}_i$ and scalars a_i, b_i such that A is reduced to tridiagonal form. The Lanczos process works as follows:

$\beta_1 \mathbf{v}_1 = \mathbf{b}$
while $i < i_{max}$ **do**
  $\mathbf{w}_i = A\mathbf{v}_i - \beta \mathbf{v}_i$
  $\alpha_i = \mathbf{v}_i^T \mathbf{w}_i$
  $\beta_{i+1}\mathbf{v}_{i+1} = \mathbf{w}_i - \alpha_i \mathbf{v}_i$
end

Algorithm 5: The Lanczos Process

Where $\mathbf{v}_0 = 0$ and β_i is chosen such that $\|\mathbf{v}_i\| = 1$ $(i > 0)$. Then after k iterations we have

$$AV_k = V_k T_k + \beta_{k+1}\mathbf{v}_{k+1}\mathbf{e}_k^T. \tag{5.21}$$

With $T_k = \text{tridiag}(\beta_i, \alpha_i, \beta_{i+1})$ and $V_k = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k]$.

When the Lanczos process is applied to the least-squares problem

$$\min \|\mathbf{b} - A\mathbf{x}\| \tag{5.22}$$

we can simplify the steps to:

$\beta_1 \mathbf{v}_1 = \mathbf{b}$
$\alpha_1 \mathbf{v}_1 = A^T \mathbf{u}_1$
while $i < i_{max}$ **do**
  $\beta_{i+1}\mathbf{u}_{i+1} = A\mathbf{v}_i - \alpha_i \mathbf{u}_i$
  $\alpha_{i+1}\mathbf{v}_{i+1} = A^T \mathbf{u}_{i+1} - \beta_{i+1}\mathbf{v}_i$
end

Algorithm 6: Bidiagonalisation

Where α and β are scalars of the bidiagonal matrix B_k, $\mathbf{u}$ of matrix U and $\mathbf{v}$ of matrix V

$$B_k = \begin{bmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \ddots & \ddots & \\ & & \beta_k & \alpha_k \end{bmatrix}, \tag{5.23}$$

$$U_k = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_k], \tag{5.24}$$

$$V_k = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k]. \tag{5.25}$$

The quantities generated by the bidiagonalisation algorithm algorithm 6 can be used to solve the least-squares problem Equation 5.22. A few additional variables need to be defined to achieve that goal. Let

$$\mathbf{x}_k = V_k \mathbf{y}_k, \tag{5.26}$$

$$\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k, \tag{5.27}$$

$$\mathbf{t}_{k+1} = \beta_1 \mathbf{e}_1 - B_k \mathbf{y}_k. \tag{5.28}$$

After some rewriting it follows that

$$\mathbf{r}_k = U_{k+1} \mathbf{t}_{k+1}. \tag{5.29}$$

Since the goal is to minimize $\|\mathbf{r}_k\|$, and because U_{k+1} is bounded and orthonormal, we should choose $\mathbf{y}_k$ such that we minimize $\|\mathbf{t}_{k+1}\|$ or the following associated least-squares problem

$$\min \|\beta_1 \mathbf{e}_1 - B_k \mathbf{y}_k\| \tag{5.30}$$

Which is the basis for the LSQR algorithm. LSQR solves Equation 5.30 with QR decomposition for B_k [2]. Where Q is a orthogonal matrix and R an upper triangle matrix. The main steps of the LSQR algorithm can be summarised into the following:

Input : $M \in \mathbb{R}^{m \times n}, \mathbf{x}_0 \in \mathbb{R}^n, \mathbf{b} \in \mathbb{R}^n$
Initialisaton
$\beta_1 \mathbf{u}_1 = b$, $\alpha \mathbf{v}_1 = A^T \mathbf{u}_1$, $\mathbf{w}_1 = \mathbf{v}_1$, $\mathbf{x}_0 = 0$, $\hat{\phi}_1 = \beta_1$, $\hat{\rho}_1 = \alpha_1$
while $i < i_{max}$ ***AND*** $\frac{\|\mathbf{r}_i\|}{\|\mathbf{b}\|} > \varepsilon$ **do**
  $\beta_{i+1} \mathbf{u}_{i+1} = A\mathbf{v}_1 - \alpha_1 \mathbf{u}_1$
  $\alpha_{i+1} \mathbf{v}_{i+1} = A^T \mathbf{u}_{i+1} - \beta_{i+1} \mathbf{v}_1$
  $\rho_1 = \sqrt{\hat{\rho}_1^2 + \beta_{i+1}^2}$
  $c_1 = \frac{\hat{\rho}_1}{\rho_1}$
  $s_1 = \frac{\beta_{i+1}}{\rho_1}$
  $\theta_{i+1} = s_i \alpha_{i+1}$
  $\hat{\rho}_{i+1} = -c_i \alpha_{i+1}$
  $\phi_i = c_i \hat{\phi}_i$
  $\hat{\phi}_{i+1} = s_i \hat{\phi}_i$
  $\mathbf{x}_i = \mathbf{x}_{i+1} + \frac{\phi_i}{\rho_i} \mathbf{w}_i$
  $\mathbf{w}_{i+1} = \mathbf{v}_{i+1} - \frac{\theta_{i+1}}{\rho_i} \mathbf{w}_i$
end

Algorithm 7: LSQR

5.4. Stopping Criterion

To stop Conjugate Gradient, CGNE or LSQR from iterating we have to use a stopping criterion. A good choice of stopping criterion balances the quality of the

criterion with the computation cost of the criterion. To satisfy this balance we use

$$\frac{\|\mathbf{r}_i\|}{\|\mathbf{b}\|} < \varepsilon \tag{5.31}$$

with $\varepsilon = 10^{-20}$. This stopping criterion scales well with the size of the problem. The norm of $\mathbf{b}$ only needs to be computed once, and the norm of $\mathbf{r}$ with each iteration. The computation cost of the norm is low, resulting in a low cost stopping criterion.

The stopping criterion $\varepsilon = 10^{-20}$ proved to be too low for good accuracy. Later we changed it to be $\varepsilon = 10^{-10}$.

5.5. Radon's Conjugate Gradient Algorithm

In this section we investigate the Conjugate Gradient algorithm that is used by Radon. The pseudo-code of the algorithm that is used by Radon can be found in Figure 5.1 (from [1]). This algorithm is equal to the preconditioned Conjugate Gradient algorithm as stated in algorithm 3.

Algoritme 7: Theoretisch **SMRCG**

1: $x = x_0$ // x_0 is het startpunt
2: $r = b - Mx$ // de residuen
3: $z = D^{-1}r$ // geschaald residu
4: $p = z$
5: $\widehat{\gamma} = z'r$ // totale geschaalde residu
6: **while** $r \neq 0$ **do**
7: $q = Mp$
8: $\alpha = \frac{\widehat{\gamma}}{p'q}$
9: $x = x + \alpha p$
10: $r = r - \alpha q$
11: $z = D^{-1}r$
12: $\gamma = z'r$
13: $\beta = \frac{\gamma}{\widehat{\gamma}}$
14: $p = z + \beta p$ // Merk op: niet $p = z + \beta q$ zoals in Lector TO
15: $\widehat{\gamma} = \gamma$
16: **end while**

Figure 5.1 *The Conjugate Gradient method used by Radon*

The programmed version of Conjugate Gradient that is used by Radon also follows these exact steps. Meaning there is no problem in the programming of Conjugate Gradient. The only difference in the code that was found is in the use of the preconditioner. More information on the different preconditioner van be found in section 5.6.

5.6. Difference in the preconditioner

During the setup to mimic the algorithm and calculations of Radon to verify if the method works there was a difference in results. Further investigation determined that this is caused by the usage of a different preconditioner. First we will recall the diagonal preconditioner that is used.

$$N\mathbf{y} = \mathbf{f} \tag{5.32}$$

where

$$N = D^{-1}MD^{-1},$$
$$\mathbf{y} = D\mathbf{x},$$
$$\mathbf{f} = D^{-1}\mathbf{b}.$$

Where the D_{ii} was calculated with

$$D_{ii} = \sqrt{M_{ii}}.$$

In the calculation of the preconditioner Radon calculates with

$$D_{ii} = M_{ii}$$

instead. This difference in preconditioning is in this case preferred over the originally stated preconditioning. That is because the new preconditioner works best for diagonal dominant matrices. We can check if a matrix is diagonally dominant with the following equation

$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}|. \tag{5.33}$$

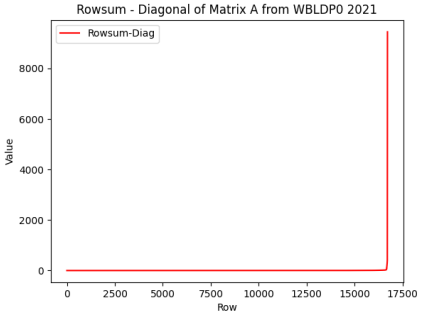
If Equation 5.33 holds for all diagonal elements of matrix A then A is diagonally dominant. By subtracting $\sum_{j \neq i} |a_{ij}|$ from both sides we can check for each diagonal element

$$|a_{ii}| - \sum_{j \neq i} |a_{ij}| \geq 0.$$

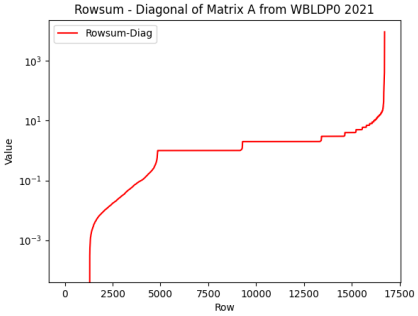
This property can be easily verified and the results are plotted in Figure 5.2. Since there are no negative values we can conclude that the matrix is diagonally dominant. Which makes the use of this preconditioner a better option than the one stated in the technological manual. Changing from $D_{ii} = \sqrt{M_{ii}}$ to $D_{ii} = M_{ii}$ will also greatly reduce the number of iterations and time needed to find a solution.

5.6.1. Calculating with updated preconditioner

After using the new preconditioner we can take another look at the convergence of the residue. Figure 5.3a shows that the convergence of Conjugate Gradient is much quicker than we have seen before in for example Figure 5.3b.

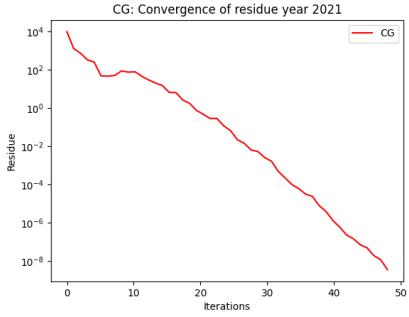


(a) *Linear view of diagonal dominance in matrix A.*

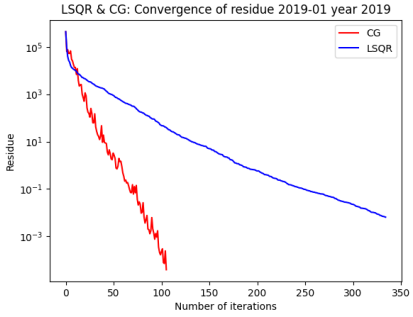


(b) *logarithmic view of diagonal dominance in matrix A.*

Figure 5.2 *Visualisation of $|a_{ii}| - \sum_{j \neq i} |a_{ij}| \geq 0$ for matrix A which is used by WBLDP0 in calculation year 2021 of case 2019-01.*



(a) *With updated preconditioner*



(b) *With old preconditioner.*

Figure 5.3 *Convergence of residue for the Conjugate Gradient calculation of $A\mathbf{x} = \mathbf{b}$ with the updated preconditioner and the old preconditioner.*

The used preconditioner is almost optimal for the forecasting problem. The best preconditioner would transform the problem such that it can be solved in one iteration but that is equal to solving the problem. Leaving little to improve on on that regard.

6. Investigating a failing forecasting case

Radon case 2019-01 is an international student test case with absolute control values. This case does not converge on a solution during the usage of Conjugate Gradient. What is observed is that the residue of the solution does not converge. Causing the algorithm to keep repeating the same Conjugate Gradient loop over and over again. We will investigate the linear system and the process of calculating the solution and determining why Conjugate Gradient does not converge.

6.1. Investigating the eigenvalues

We start the investigation by calculating the eigenvalues of matrix M in linear system $M\mathbf{x} = \mathbf{b}$, with $M = AA^T$. One thing that can cause Conjugate Gradient to not converge is that matrix M is singular. Let us first define the determinant of matrix A.

Definition 6.1.1 *Determinant*
The determinant of a matrix $A \in \mathbb{R}^{n\times n}$ is the product of its eigenvalues.
$\det(A) = \lambda_1\lambda_2 \dots \lambda_n$

Definition 6.1.2 *Singular matrix*
A matrix $A \in \mathbb{R}^{n\times n}$ is singular if and only if its determinant is 0 ($\det(A) = 0$).

From this we can conclude that a square matrix A is a singular matrix if and only if it has at least one eigenvalue 0. We can use the singularity of matrix A to say something about the convergence of the linear system. In subsection 5.1.2 we have covered the convergence properties of Conjugate Gradient. Showing that the convergence of the error $\|\mathbf{e}_i\| = \|\mathbf{x} - \mathbf{x}_i\|$ obeys the following inequality,

$$\|\mathbf{e}_i\|_A \leq 2\left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1}\right)^i \|\mathbf{e}_0\|. \tag{6.1}$$

When an eigenvalue of matrix A is 0 then the condition number $\kappa_2(M)$ can not be defined. When this happens we set the condition number $\kappa_2(M) = \infty$. If we use this in the convergence Inequality (Equation 6.1) we see that convergence is not guaranteed.

6.1.1. Calculating the eigenvalues of case 2019-01

Knowing that an eigenvalue 0 can cause Conjugate Gradient to diverge we start by calculating the eigenvalues of matrix M that is used in the linear system from case 2019-01. Since Conjugate Gradient is called by the algorithm multiple times we take a look at the first use of Conjugate Gradient. We can also determine the moment when the method diverges and look closer to that linear system.

In the first case, with the initial problem we have and A matrix with dimension (12021, 81249). This matrix consists only of 1's, -1's and 0's. Matrix A holds all the information on the constraints on the education matrix. Additionally it contains the information of the absolute control equations. We will be calculating the eigenvalues of matrix $M = AA^T$ as this is used to solve the problem. Calculating the eigenvalues of this matrix gave the following information:

- $\lambda_{\max} = 9848$,
- The average eigenvalue is 1 or 2,
- $\lambda_{\min} = 0$.

Since $\lambda_{\min} = 0$ we have at least one eigenvalue 0. Upon further inspection this is also the only eigenvalue 0. An eigenvalue 0 is not a guarantee that the problem does not converge.

Next we have a look at the eigenvalues of the linear system of case 2019-01 which does not converge to a solution. The dimensions of matrix A are hugely different with (16698, 81249), something we investigate in section 6.4. The maximum and minimum eigenvalue are both 0. So each eigenvalue is 0. This looks like the main reason that Conjugate Gradient can not find a solution.

6.1.2. Comparing case 2019-01 to case 2020-00

We compare the eigenvalues of case 2019-01 with the eigenvalues of a case that does converge to a solution. We call this case 2020-00. Matrix A in case 2020-00 has dimension (12234, 98953). We again calculate the eigenvalues of matrix $M = AA^T$.

- $\lambda_{\max} = 524$,
- The average eigenvalue is 1 or 2,
- $\lambda_{\min} = 0$.

The minimum eigenvalue is again 0. With only one occurrence of eigenvalue 0. So even in a working case there appears to be a singular matrix. In Table 6.1 we have a small overview of the three cases we studied.

Case	2019-01	2019-01 ∞-loop	2020-00
$\lambda_{\max}$	9848	0	524
$\lambda_{\min}$	0	0	0
κ_2	∞	∞	∞
A **dimensions**	(12021, 81249)	(16698, 81249)	(12235, 98953)

Table 6.1 *Overview of eigenvalues*

6.2. Cause of eigenvalue 0

It is quite likely that the 0 eigenvalue causes Conjugate Gradient to diverge here. Because of that we will try to determine the cause of this 0. We list two equivalent causes of a matrix being singular.

- There is linear dependence in the columns of matrix A.
- Matrix A does not have full rank. (rank(A)$\neq n$).

Definition 6.2.1 *Rank of matrix A*
The rank of matrix A (rank(A)) is equal to the dimension of the vector space generated by its columns.

Linear dependence will be hard to check in large matrices. We can check each row in the matrix and see if there is linear dependence in the matrix. This process will take a lot of time so we assume that there are no linear dependencies created during the construction of matrix A.

Calculating the rank of A will be easier. One can use Gaussian Elimination to reduce matrix A to an upper triangle form called row echelon. The vector space of the row echelon form is easily determined due to its upper triangle form. Leading to an easy calculation of rank(A). Since Gaussian elimination is widely used in numerical methods as a direct solving method known as LU-factorization ([12]), we will investigate this first.

6.2.1. LU-factorization

Gaussian Elimination or LU-factorization is used in direct solution method to solve linear systems. Most direct methods first factorize the original matrix A,

$$A = LU, \tag{6.2}$$

into a product of an upper triangle matrix U and a lower triangle matrix L with diagonal elements set to one. Matrix U will be in row echelon form and have the same rank as A. We calculate the rank of U and find that rank$(U) = 12020 \neq 12021$ as expected. This means that matrix U and matrix A are singular with only one eigenvalue 0.

6.2.2. Row of zeroes by mistake

One of the easiest ways to create an eigenvalue 0 in a matrix is to have a row or column full of zeroes. First we calculate the absolute sum of each row and then check if any of them are 0. It appears the last row of the matrix is full of zeroes. Upon reviewing my own code it was noticed that matrix A was one row larger than it should have been. Adding a row of zeroes at the end and thus creating a singular matrix. We remove this mistake from the matrix and recalculate the eigenvalues in the following section.

6.2.3. Recalculate the eigenvalues

Since there was a mistake in the matrix we have to recalculate the eigenvalues with the correct matrix. Table 6.2 has the minimum and maximum eigenvalues for all three cases.

Case	**2019-01**	**2019-01 ∞-loop**	**2020-00**
$\lambda_{\max}$	9848	0	524
$\lambda_{\min}$	0.2679	0	0.38
κ_2	36753.3683	∞	1372.218
A **dimensions**	(12020, 81248)	(16709,81248)	(12234, 98956)

Table 6.2 *Overview of eigenvalues*

For both case 2019-01 and 2020-00 not a lot has changed because only the eigenvalue 0 was removed. Case 2019-01 ∞-loop is a weird case. Calculating the eigenvalues at this time leads to many different results. One time it resulted in complex eigenvalues and another in all zeroes. This may be a result of round-off errors. We plot the eigenvalues of both 2019-01 and 2020-00 on their first Conjugate Gradient usages in Figure 6.1. The eigenvalues are almost identical. We can state that all the eigenvalues of case 2019-01 and 2020-00 are positive.

6.3. Separate calculation with Conjugate Gradient and LSQR

To check if the issue lies with the Conjugate Gradient method of Radon we will solve the linear system with a separate Conjugate Gradient and LSQR method. We apply LSQR and Conjugate Gradient to the linear problems with selection and weight. Table 6.3 shows how many iterations are needed for a method to reach convergence. LSQR does not converge with the given maximum number of iterations. The residuals for both methods are plotted in Figure 6.2 for both the initial use of Conjugate Gradient and the infinite loop system.

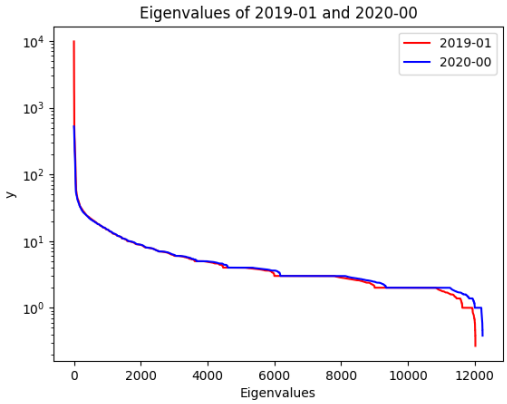


Figure 6.1 *Eigenvalues of both 2019-01 and 2020-00 on their first Conjugate Gradient usage.*

Case	2019-01	2019-01 ∞-loop	2020-00
LSQR iterations	1000 (max)	1000(max)	1000 (max)
CG iterations	430	1000(max)	420

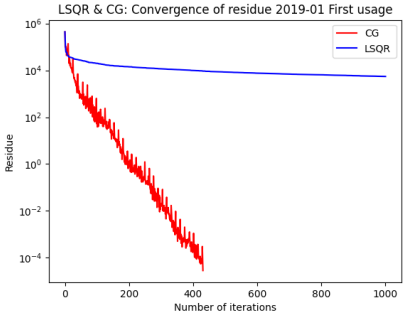
Table 6.3 *Overview of running Conjugate Gradient and LSQR*

An increase in maximum iterations does not allow LSQR to find a solution or Conjugate Gradient to find a solution in the infinite loop. We can see the same happening in case 2020-00 in Figure 6.3.

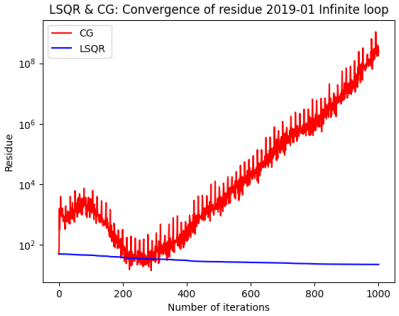
6.3.1. Do control equations break the problem?

We noticed that a different A matrix is imported for different calculation years. On further inspection we have seen that after matrix A is imported the control equations are added. This increases the number of columns and rows of the matrix by 10. Before we dive deeper into that difference we will first check if we can calculate the solution of the linear systems $A\mathbf{x} = \mathbf{b}$ before any control equations are added.

We can conclude from Figures Figure 6.4 and Figure 6.5 that the problem does not originate from adding the control equations to the linear system. The addition did cause Conjugate Gradient to be more erratic, but does not disturb its convergence. So in this case the problem does not occur in the calculation of $\mathbf{y}_*$ in minimization problem Equation 4.3. This means that the problem occurs in problem Equation 4.4 or in the optimization of $\mathbf{z}_*$.



(a) *Convergence of residue for 2019-01 First usage*



(b) *Convergence of residue for 2019-01 infinite loop*

Figure 6.2 *The convergence of residue of 2019-01 with both LSQR and Conjugate Gradient. These are both with selection and weight. LSQR does not converge to 0 in a reasonable amount of time while Conjugate Gradient does for the initial problem. In the infinite loop Conjugate Gradient is unable to find a solution while LSQR has the same issues as in the initial problem.*

6.4. Different sizes for A for different calculation years

In the initiation phase Radon adds the control sequences to A. adding both rows and columns to the matrix. However, the added rows and columns are only equal to the number of constraints that are imposed, in this case 10 constraints. Which does not explain the large increase that was observed earlier. What does is the fact that Radon solves each year individually. Importing a different matrix A for each calculation year. We checked matrix A for the first use of SMRCG (Scaled Minimum Residual Conjugate Gradients) in year 2019 and found the sizes which we note below. When the calculations are underway matrix A no longer changes in dimension. If we look to the calculation year 2020 there is a huge difference in the number of rows in matrix A. For each year up until a specified year, a new matrix A is imported for that year. We give an example for different dimensions for different calculation years:

```
Calculation year 2019:
Columns of A:   81238
Rows of A:      12010

Calculation year 2020:
Columns of A:   81238
Rows of A:      15922
```

From this we can conclude that the difference in dimensions originates from the input.

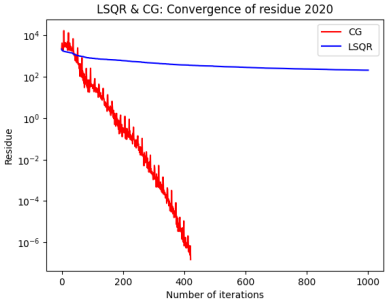
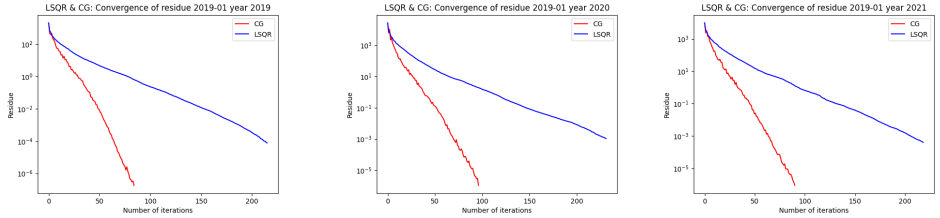
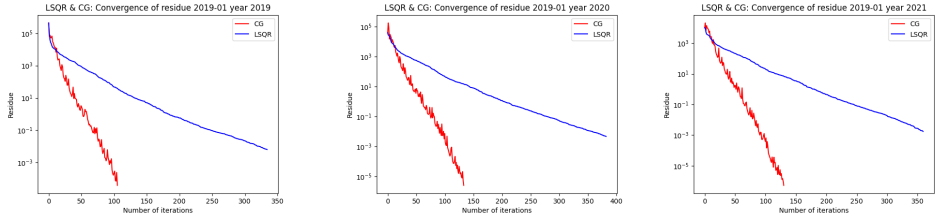


Figure 6.3 *The convergence of residue of 2020-00 with both LSQR and Conjugate Gradient. These are both with selection and weight. Again, Conjugate Gradient converges while LSQR does not.*



(a) *Calculation year 2019.* ***(b)*** *Calculation year 2020.* ***(c)*** *Calculation year 2021.*

Figure 6.4 *Convergence of residue with matrix A as imported for year 2019, 2020 and 2021. This linear system has been solved without adding control equations to A*



(a) *Calculation year 2019.* ***(b)*** *Calculation year 2020.* ***(c)*** *Calculation year 2021.*

Figure 6.5 *Convergence of residue with matrix A as imported for year 2019, 2020 and 2021. In this case we did add control equations to A.*

6.5. Optimizing the initial solution

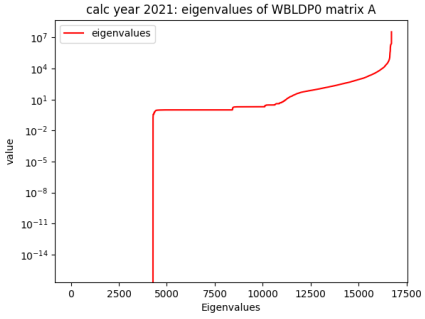
We will continue the investigation by optimizing the initial solution to also fit the constraints. Calculating a solution to the following linear system:

$$\begin{gathered} \min_{\mathbf{z}} \|\mathbf{z}\|_W \\ \textbf{subject to} \\ A\mathbf{z} = \mathbf{b} - A\mathbf{a} - A\mathbf{y}_* \\ S^*(\mathbf{z} + \mathbf{y}_* + \mathbf{a}) \geq 0 \\ 0 \leq W \leq \infty \end{gathered} \tag{6.3}$$

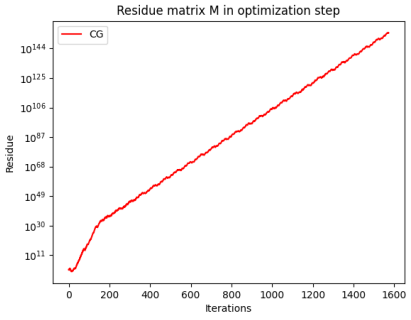
This system is again solved with the conjugate gradient method. The starting vector of the conjugate gradient algorithm used in the optimisation part is set to $\mathbf{x}_0 = \max(0, -(\mathbf{a} + \mathbf{y}_*))$. Containing only the negative values of $\mathbf{y}_* + \mathbf{a}$. The values which are not compatible with the constraint $S^*(\mathbf{y}_* + \mathbf{a}) \geq 0$.

6.6. The big issue with selection

At this point the algorithm only takes into account the rows of matrix A for which $\mathbf{x}_0^i$ is non-zero with selector $\hat{S}$. The selector $\hat{S}$ selects around half of the columns of matrix A. Calculating with matrix $M = A\hat{S}W^{-1}A^T$. Figure 6.6a shows the eigenvalues of the used matrix M. In the case of 2019-01 matrix M will have around 4600 rows full of zeroes. Having so many zero eigenvalues is a cause that conjugate gradient will not converge to a solution as shown in Figure 6.6b. This is also the point at which Radon enters an infinite loop.



(a) *The eigenvalues of matrix $M = ASW^{-1}A^T$.*



(b) *The residue of Conjugate Gradient in the optimisation step.*

Figure 6.6 *The eigenvalues of matrix M and the residue of the Conjugate Gradient method applied to the linear system $M\mathbf{x} = \mathbf{b}$.*

6.7. Example of selection

In this section we cover how selection is applied to matrix M.

6.7.1. Visualization of selection

Let us first show a visualization of how the selection is applied in Figure 6.7.

We will apply a selection to the diagram in Figure 6.7 and select only nodes $[3, 4, 5, 6]$, encircled by the orange square. Selecting only the elements of the ed-

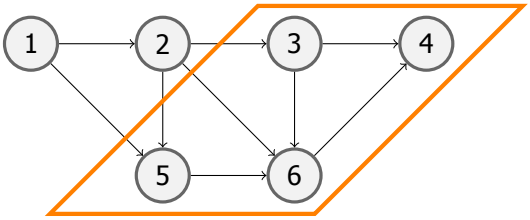


Figure 6.7 *A visualization of an example education matrix and its movement. In this case* $\mathbf{x} \in \mathbb{R}^{(6\times6)}$*. The movement from an origin category to a destination category is shown with an arrow. We apply a selection to the categories encircled by orange.*

ucation matrix which are in both the selected row and the selected column of the education matrix. For category 3 this is $[\mathbf{x}_{33}, \mathbf{x}_{34}, \mathbf{x}_{35}, \mathbf{x}_{36}, \mathbf{x}_{43}, \mathbf{x}_{53}, \mathbf{x}_{63}]$.

6.7.2. The issue with selection

The divergence issue stems from the fact that a selector is used by Radon before applying conjugate gradient. In this section we will have a closer look at how the selector is used; if it does what it is supposed to; and if the result is what is expected. We show all of these with a simple example. Let A and S be defined as follows:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}, \qquad S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{6.4}$$

We want to select only the first and second column in matrix A corresponding to the first and second variable. Then according to the current method of applying the selector in $M = ASA^T$:

$$M = ASA^T \tag{6.5}$$

$$= \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix} \tag{6.6}$$

$$= \begin{bmatrix} 5 & 17 & 29 \\ 17 & 61 & 105 \\ 29 & 105 & 181 \end{bmatrix}. \tag{6.7}$$

And if we have a selector selecting all columns with diag$(S) = [1, 1, 1, 1]$

$$M = ASA^T \tag{6.8}$$

$$= \begin{bmatrix} 30 & 70 & 110 \\ 70 & 174 & 278 \\ 110 & 278 & 446 \end{bmatrix} \tag{6.9}$$

Showing that the selection does have an effect on matrix M. We will continue with this example by applying Conjugate Gradient to the linear system $ASA^T\mathbf{u} = \mathbf{b}$ with $\mathbf{b} = [1, 2, 3]^T$. With selector diag$(S) = [1, 1, 1, 1]$ and inital solution $\mathbf{u}_0 = [0, 0, 0]^T$ we have conjugate gradient calculate a solution $\mathbf{u}$. Then the solution $\mathbf{x}$ for $A\mathbf{x} = \mathbf{b}$ can be calculated by

$$SA^T\mathbf{u} = \mathbf{x} = [-0.05, 0.025, 0.1, 0.175]. \tag{6.10}$$

The same can be done for diag$(S) = [1, 1, 0, 0]$. Solving linear system $ASA^T\mathbf{u} = \mathbf{b}$ with Conjugate Gradient gives the following solution for $\mathbf{x}$

$$SA^T\mathbf{u} = \mathbf{x} = [-0.5, 0.75]. \tag{6.11}$$

Giving a solution that ignores the last two variables of $\mathbf{x}$. Which is the same solution as calculating with the following matrix A without selection.

$$\hat{A} = \begin{bmatrix} 1 & 2 \\ 5 & 6 \\ 9 & 10 \end{bmatrix}. \tag{6.12}$$

Applying the selection thus calculates a solution to the smaller linear system $\hat{A}\mathbf{x} = \mathbf{b}$. At the end of the calculation Radon merges the values of $\mathbf{x}$ that were not selected to the solution vector Equation 6.11. In this example the final solution would be

$$\mathbf{x}_* = \begin{bmatrix} -0.5 \\ 0.75 \\ 0.1 \\ 0.175 \end{bmatrix}. \tag{6.13}$$

6.8. How can the selection process go wrong?

In the example of section 6.7 we see nothing that could lead the conjugate gradient method to diverge. The issue that caused Conjugate Gradient to diverge was rows or columns full of zeroes in matrix M. The only way that is possible if the selector selects only the values of zero in a row or column of matrix A.

6.8.1. Zero row and column

A row or column full of zeroes is a possibility when applying a selector. We continue with a similar example as in the previous section 6.7 to show how. We

define A and S as

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 0 & 0 & 11 & 12 \end{bmatrix}, \qquad S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{6.14}$$

Calculating matrix M

$$M = ASA^T \tag{6.15}$$

$$= \begin{bmatrix} 5 & 17 & 0 \\ 17 & 61 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{6.16}$$

As we can see matrix M now has a row and a column filled with zeroes. An application of Conjugate Gradient on system $M\mathbf{x} = \mathbf{b}$ will not give a solution since M is singular.

6.8.2. Linear dependence

Another possibility is that matrix M can contain linear dependent rows after the selection.

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 2 & 4 & 11 & 12 \end{bmatrix}, \qquad S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{6.17}$$

Calculating matrix M

$$M = ASA^T \tag{6.18}$$

$$= \begin{bmatrix} 5 & 17 & 10 \\ 17 & 61 & 34 \\ 10 & 34 & 20 \end{bmatrix}. \tag{6.19}$$

Applying Conjugate Gradient to the linear system $M\mathbf{x} = \mathbf{b}$ with matrix M from Equation 6.19 does not converge due to the linear dependence in rows one and three. If we bring this back to the basics then this becomes more apparent.

Take for example

$$x_1 + x_2 = 1 \tag{6.20}$$

$$2x_1 + 3x_2 = 4 \tag{6.21}$$

The solution to this system is $x_1 = -1, x_2 = 2$. If we select only the first variable we end up with the following system

$$x_1 + 0x_2 = 1 \tag{6.22}$$

$$2x_1 + 0x_2 = 4, \tag{6.23}$$

which has no solution.

6.9. Alternatives selection process

Since the current use of selection can result in singular matrices, we will attempt to use alternative methods to using selection. The first alternative is to additionally apply a selector to matrix M. An example will show if that works out. Another attempt is to not calculate matrix M and to numerically solve $Ax = b$ with CGNE, whilst removing the unselected columns from matrix A.

6.9.1. Additional selection on M

In this section we will also apply the selection to matrix M. Selecting only the rows that correspond with the selected column. This allows us to also drop any zero rows or columns that could cause issues in Conjugate Gradient. One downside is that it could also drop non-zero and non-linear dependent rows or columns.

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}, \quad S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad M = \begin{bmatrix} 5 & 17 & 29 \\ 17 & 61 & 105 \\ 29 & 105 & 181 \end{bmatrix}. \tag{6.24}$$

With solution $\mathbf{x} = [-0.5, 0.75, 0, 0]^T$.

Then we select only the rows corresponding with the chosen columns:

$$M = \begin{bmatrix} 5 & 17 \\ 17 & 61 \end{bmatrix}, A = \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} \tag{6.25}$$

Calculating the linear system $M\mathbf{u} = \mathbf{b}$ with $\mathbf{u}_0 = [0, 0]^T$ and $\mathbf{b} = [1, 2]^T$ gives the following solution.

$$\mathbf{u} = [0.10189, 0.02651]^T \tag{6.26}$$

$$\mathbf{x} = A^T\mathbf{u} = \begin{bmatrix} 1 & 5 \\ 2 & 6 \end{bmatrix} [0.10189, 0.02651]^T = [0.24242, 0.37878]. \tag{6.27}$$

The solution of this selected system does not achieve the desired results. Even if the results of this linear system was correct there could be another issue when the selector selects the last column with $S = [0, 0, 0, 1]$. because then what is selected in matrix M? This does not lead to a replacement method.

6.9.2. Use CGNE and remove unselected columns

The use of CGNE instead of Conjugate Gradient allows us to apply the selection to matrix A instead of to matrix M. In addition to this selection we will also delete the columns that are not selected. We again use the same example:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}, \qquad S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{6.28}$$

After applying selection and deleting the unselected columns we calculate the following linear system:

$$\begin{bmatrix} 1 & 2 \\ 5 & 6 \\ 9 & 10 \end{bmatrix} x = b. \tag{6.29}$$

Which sadly has the same issues as the current use of the selection. We can end up with a singular matrix because of a row full of zeroes or linear dependence.

6.9.3. Conclusion

In subsection 6.9.1 and subsection 6.9.2 we tried two alternative selection methods. Neither method gives a proper solution to the linear system or gives a solution to a totally different linear system, unrelated to the original system. Meaning these are not able to replace the current method of selection.

7. Conclusion and recommendations

7.1. Conclusions

In this thesis we examined the numerical Conjugate Gradient algorithm that is used in the prediction of the number of students in the Netherlands. The addition of international students to the education matrix caused issues with the test cases and Radon was unable to find a solution. In this thesis we also investigated one of the test cases to determine what caused the issue.

We determined that the Conjugate Gradient method that was implemented for the predictions was correctly programmed. The only difference that was found was that the diagonal preconditioner that was used was improved upon without changing the documentation in the technological manual ([1]). The currently used preconditioner proved to be a better choice because the used matrices are primarily diagonal dominant matrices.

After thoroughly investigating the algorithm to solve the forecasting problem and the international student test case we came to the conclusion that there is an issue with the usage of a selector. The current usage of the selector in $ASA^T = M$ can result in a singular matrix M. A linear system with a singular matrix does not have a solution. So Conjugate Gradient or any other algorithm is unable to find a solution. We have tried two methods to replace the current way of selecting but both of them ended up with singular matrices or solutions that can not be used for the problem.

7.2. Recommendations for further research

The work represented in this thesis shows that the issue lies within the application of the selector. To further research how this issue can be prevented we recommend an investigation in a few topics.

- **New selection process**
  The issue that was found is that the current way the selector is used in the linear system causes a singular matrix and thus a solution can not be found. We had a look at two other methods of applying a selection. Neither of these two methods led to a proper solution or working selection. It is recommended to look at possible different selection processing. If this turns out to be impossible then the following is suggested.
- **Prevent using selection**
  Another possibility of solving the issue is by preventing the use of selection. Currently Radon solves the problem in multiple steps. First finding an initial

solution without taking into account the constraints that are imposed on the problem. It should be possible to solve the forecasting problem from the start with all of these constraints imposed on the problem. Preventing the use of the selector in later steps.

Bibliography

[1] H. Adriaens, K. de Vos, and P. Fontein. *Technisch ontwerp rekenmodule Radon*. 2010.

[2] Gene Golub. "Numerical methods for solving linear least squares problems". In: *Numerische Mathematik* 7.3 (1965), pp. 206–216.

[3] Gene H Golub and William Kahan. "Calculating the singular values and pseudo-inverse of a matrix". In: *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis* 2.2 (1965), pp. 205–224.

[4] Gene H Golub and Charles F Van Loan. *Matrix computations*. Vol. 3. JHU press, 2013.

[5] M. Heres Hoogerkamp, N. Bleijie, and C. Chiong Meza. *Historisch overzicht referentieramingen.* 2021.

[6] Directie Kennis/PSB. *Een stukje historie van de Leerlingenraming bij OCW*. 2016.

[7] B. Kuhry. *Prognoseperikelen. De voorspelkracht van prognoses van het voorzieningengebruik.* 1995.

[8] Cornelius Lanczos. *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Governm. Press Office Los Angeles, CA, 1950.

[9] Christopher C Paige and Michael A Saunders. "LSQR: An algorithm for sparse linear equations and sparse least squares". In: *ACM Transactions on Mathematical Software (TOMS)* 8.1 (1982), pp. 43–71.

[10] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[11] Taakgroep Studentenramingen. *Rhobos '81: Raming hoger beroepsonderwijs studentenaantallen 1981-1995.* 1981.

[12] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*. Vol. 50. Siam, 1997.

[13] C. Vuik and D. Lahaye. *Scientific computing, Lecture notes for WI4201*. 2012.

A. List of Variables

- *Origin vector* $H \in \mathbb{R}^n$ – In the origin vector we record the number of students in each education category before they move to the destination vector. This vector is the same as the destination vector of the year before.
- *Destination vector* $B \in \mathbb{R}^n$ – In the destination vector we record the number of students that have to end up in each education category.
- *Education matrix* $x \in \mathbb{R}^{m \times n}$ – The education matrix contains the movement that happens in education each year. The students that reside in the origin categories will travel to the destination categories according to the values in the education matrix.
- *Absolute constraint* – An absolute constraint on the education matrix is a predetermined control value on how many students end up in a particular destination category. e.g. $x_{11} + x_{21} = 10$ if we want to impose a value of 10 on destination category 1.
- *Relative constraint* – A relative constraint on the education matrix is a predetermined percentage on students leaving an origin category. This constraint will force a percentage of students leaving a category to go to a chosen category. e.g. $\frac{x_{12}}{x_{11}+x_{12}} = 0.1$ is a relative constraint such that the stream from category 1 to category 2 is 10 % of the whole stream of students leaving category 1.

B. Solving examples of linear systems

B.1. Conjugate Gradient

In this section we cover some simple examples to check if our implemented Conjugate Gradient algorithm with preconditioning works. We will have a look at simple square diagonal matrices for Conjugate Gradient and a preconditioned Conjugate Gradient. We will expand the square matrices to rectangular $A \in \mathbb{R}^{m \times n}$ with $m < n$ to test CGNE. After these simple matrices we will do the same for the 1D Poisson equation:

$$-\frac{\mathrm{d}^2 u}{\mathrm{d}x^2} = f \tag{B.1}$$

The exact solution to this equation can be easily calculated which allows us to compare it with the computed solution of Conjugate Gradient.

B.1.1. CG: Square diagonal matrix A

Let $A \in \mathbb{R}^{10 \times 10}$, $\mathbf{x} \in \mathbb{R}^{10}$, $\mathbf{b} \in \mathbb{R}^{10}$. We want to have a simple diagonal matrix A and a simple vector b so we can easily see if the computed vector $\mathbf{x}$ from the Conjugate Gradient algorithm is correct. For A we pick a diagonal matrix with the following values on the diagonal:

$$\mathrm{diag}(A) = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$$

The rest of the values of A are zeros. For $\mathbf{b}$ we take the same values as diag(A):

$$\mathbf{b} = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$$

This results in the following system

$$\begin{bmatrix} 1 & & & \\ & 2 & & \\ & & \ddots & \\ & & & 10 \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{10} \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ \vdots \\ 10 \end{bmatrix} \tag{B.2}$$

One can easily see that the solution of the system is the vector of ones $\mathbf{x} = \mathbf{1}$.

Computing the solution

We first compute the solution with Conjugate Gradient and then we compute it with a preconditioned Conjugate Gradient. The Conjugate Gradient algorithm

requires an initial solution vector for $\mathbf{x}$. For this we pick $\mathbf{x}_0 = \mathbf{0}$. With everything set up we can run the Conjugate Gradient algorithm and are left with the following results.

```
Example 1: Square Diagonal Matrix A
CG: Solution found after 16 iterations
x = [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Preconditioned CG: Solution found after 10 iterations
x = [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Elapsed time for Standard CG: 0.0020012855529785156 seconds
Elapsed time for Preconditioned CG: 0.0020004043579101562 seconds
```

As expected the solution is $\mathbf{x} = \mathbf{1}$. The computation time of both is also the same, but so low that we can't draw a conclusion. For an easy diagonal example such as this it is often the case that a solution is found after a number of iterations equal to $n = 10$. Notice that solution for Conjugate Gradient was found after 16 iterations, more than the expected 10. This is because we had a too tight of a stopping criterion of $\varepsilon = 10^{-20}$. After changing the stopping criterion to $\varepsilon = 10^{-10}$ we have the following results, lowering the number of iterations.

```
Example 1: Square Diagonal Matrix A
CG: Solution found after 10 iterations
x = [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Preconditioned CG: Solution found after 10 iterations
x = [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

In Figure B.1 we can see the speed of convergence for both Conjugate Gradient and preconditioned Conjugate Gradient. The residual of preconditioned Conjugate Gradient converges faster at the start but will equalize with Conjugate Gradient after n iterations.

B.1.2. CG: Square 1D Poisson equation

We use the Conjugate Gradient algorithm to find the solution to the 1D Poisson equation.

$$-\frac{\mathrm{d}^2 u}{\mathrm{d}x^2} = f \tag{B.3}$$

We can discretize the Poisson equation using central finite difference approxi-

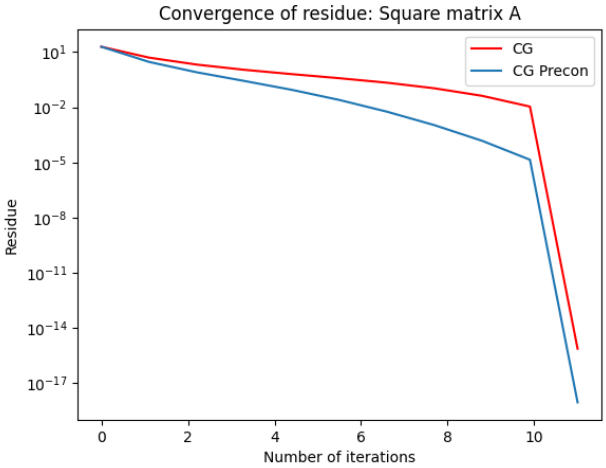


Figure B.1 *The convergence of the residual of Conjugate Gradient and preconditioned Conjugate Gradient for the system given in Equation B.2. The residual of the preconditioned system converges faster.*

mation. Using nearest neighbour we have

$$-\frac{\mathrm{d}^2 u}{\mathrm{d}x^2} = \frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} + \mathcal{O}(h^2). \tag{B.4}$$

The discretization can be represented in stencil notation, which represents the coupling of the unknowns in the left and right neighbours, by

$$\frac{1}{h^2}[-1 \quad 2 \quad -1].$$

We translate the 1D Poisson Equation B.3 into a linear system of equations using the finite difference approximation

$$A^h \mathbf{u}^h = \mathbf{f}^h. \tag{B.5}$$

Matrix A^h will represent the discretized differential operator. $\mathbf{u}^h$ the second order approximation of the solution $\mathbf{u}$. The domain on which we practise the Poisson example is $[0, 2\pi]$. This domain will be split up into N mesh elements, creating a grid consisting of $N+1$ nodes placed on equal distance from each other at $h = \frac{2\pi}{N}$. All that is left is to use a suitable $\mathbf{f}$ with an easy to calculate second derivative.

$$u = x\cos(x) \tag{B.6}$$

$$-\frac{\mathrm{d}^2 u}{\mathrm{d}x^2} = 2\sin(x) + x\cos(x) \tag{B.7}$$

The boundary conditions for this problem are Dirichelet boundary conditions

$$u(0) = 0 \tag{B.8}$$
$$u(2\pi) = 2\pi \tag{B.9}$$

Since we have Dirichlet boundary conditions we can calculate the first node and the last node separately. In addition to that we can make matrix A symmetric by translating the connections of the left-most and right most interior points to the left and right boundary point into contributions to $\mathbf{f}$. We will give the details on the system given in Equation B.5:

$$\frac{1}{h^2}\begin{pmatrix} h^2 & 0 & 0 & \dots & \dots & \dots & 0 \\ 0 & 2 & -1 & 0 & \dots & \dots & 0 \\ \vdots & -1 & 2 & -1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \dots & 0 & -1 & 2 & -1 & 0 \\ 0 & \dots & \dots & 0 & -1 & 2 & 0 \\ 0 & \dots & \dots & \dots & 0 & 0 & h^2 \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \\ u_{N+1} \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 + \alpha \\ \vdots \\ f_N + \beta \\ f_{N+1} \end{pmatrix} \tag{B.10}$$

Computing the solution

We run the Conjugate Gradient algorithm 1 and the preconditioned algorithm 3 for the 1D Poisson equation and get the following results.

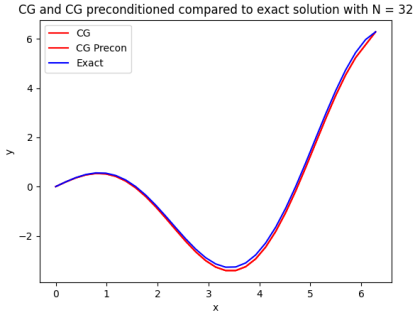
```
Example 2: Square 1D Poisson equation
N=32
CG: Solution found after 30 iterations
Preconditioned CG: Solution found after 30 iterations
N=128
CG: Solution found after 126 iterations
Preconditioned CG: Solution found after 126 iterations
```

We plot both solutions and the exact solution in Figure B.2. The computed solution is close to the exact solution from Equation B.7. Showing us that the algorithm works as intended.

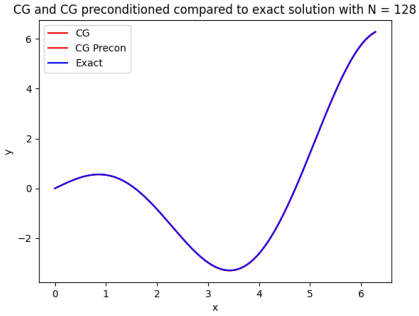
In Figure B.3 we have visualised the convergence of the residuals for $N = 32$ and $N = 128$. The last residual has been left out as there was a large jump to 10^{-10}

B.2. CGNE

In this section we will cover the same examples as in section B.1. However, we will make a slight modification to matrix A by adding extra columns consisting only

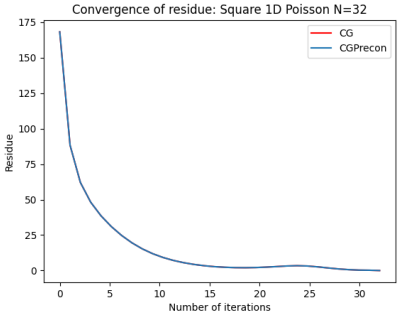


(a) *CG Solution for $N = 32$*

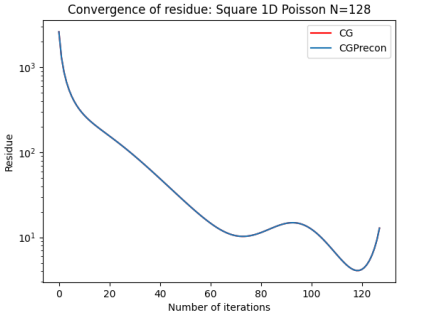


(b) *CG Solution for $N = 128$*

Figure B.2 *CG solution $\mathbf{u}$ compared to the exact solution for $N = 32$ and $N = 128$. A higher number of grid nodes has a higher accuracy.*



(a) *Convergence of residue for $N = 32$*



(b) *Convergence of residue for $N = 128$.*

Figure B.3 *The convergence of the residuals of the square 1D Poisson system on a logarithmic scale. For $N = 32$ we have a decreasing graph. The residual for $N = 128$ is no longer a decreasing function and even increasing in the end before hitting the stopping criterion.*

of zeros. With the rectangular matrix A we can test CGNE algorithm 2.

B.2.1. CGNE: Rectangular diagonal matrix A

Let $A \in \mathbb{R}^{10\times 20}$, $\mathbf{x} \in \mathbb{R}^{30}$, $\mathbf{b} \in \mathbb{R}^{10}$. We want to have a simple rectangular matrix A and a simple vector b so we can easily see if the computed vector x from the CGNE algorithm is correct. For A we pick the diagonal matrix from subsection B.1.1 with the following values on the diagonal:

$$\text{diag}(A) = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$$

We then add 10 more columns filled with a diagonal set to ones to A making it a rectangular matrix. For $\mathbf{b}$ we take the same values as diag(A):

$$\mathbf{b} = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$$

This results in the following linear system

$$\begin{bmatrix} 1 & & & & 0 & & & \\ & 2 & & & & 0 & & \\ & & \ddots & & & & \ddots & \\ & & & 10 & & & & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{20} \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ \vdots \\ 10 \end{bmatrix}. \tag{B.11}$$

The solution for this equation is $x_1 \dots x_{10} = 1$ and additionally (without loss of generality) $x_{11} \dots x_{20} = 0$.

Computing the solution

We compute the solution with CGNE algorithm 2 and the preconditioned version algorithm 4.

```
 Example 3: Rectangular diagonal matrix A
 CGNE: Solution found after 10 iterations
x = [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 CGNE Precon: Solution found after 1 iteration
x = [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

CGNE computes the expected solution in the same number of iterations as Conjugate Gradient did for the square matrix. CGNE Precon actually computes the solution in its first iteration. Making it a lot faster than the non-preconditioned CGNE and more like a direct method. When we visualise the residual in Figure B.4 we can see that the residual of the preconditioned CGNE starts a lot smaller than of CGNE and iterates below the stopping criterion after the first iteration.

B.2.2. CGNE: Rectangular 1D Poisson equation

We follow the same steps as in subsection B.1.2. Using the CGNE algorithm to find the solution to the 1D Poisson equation.

$$-\frac{\mathrm{d}^2 u}{\mathrm{d}x^2} = f \tag{B.12}$$

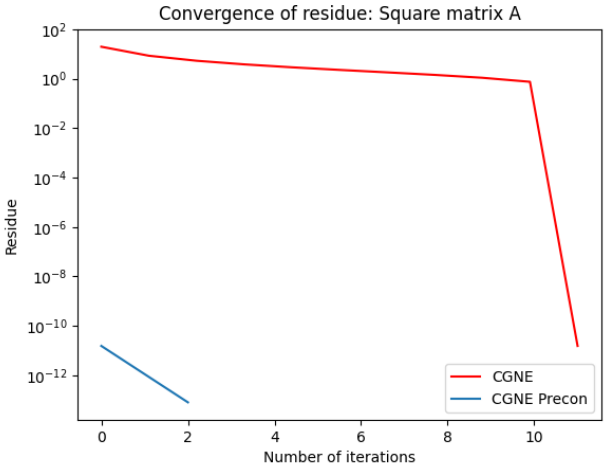


Figure B.4 *Residuals for CGNE and preconditioned CGNE on rectangular matrix A.*

Solving the linear system associates with the central discretization, adding extra columns containing zeros to A^h.

$$\begin{pmatrix} h^2 & 0 & 0 & \dots & \dots & \dots & 0 & 0 & \dots & 0 \\ 0 & 2 & -1 & 0 & \dots & \dots & 0 & \vdots & \ddots & \vdots \\ \vdots & -1 & 2 & -1 & 0 & \dots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \dots & 0 & -1 & 2 & -1 & 0 & \vdots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & -1 & 2 & 0 & \vdots & \ddots & \vdots \\ 0 & \dots & \dots & \dots & 0 & 0 & h^2 & 0 & \dots & 0 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{3N+2} \\ u_{3(N+1)} \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \\ f_{N+1} \end{pmatrix} \tag{B.13}$$

With

$$\mathbf{f} = x\cos(x). \tag{B.14}$$

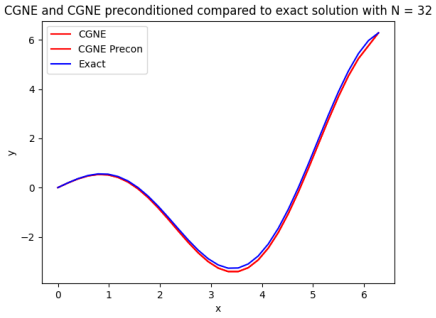
Computing the solution

The solution to the system given in Equation B.13 can be computed with the CGNE algorithm 2.

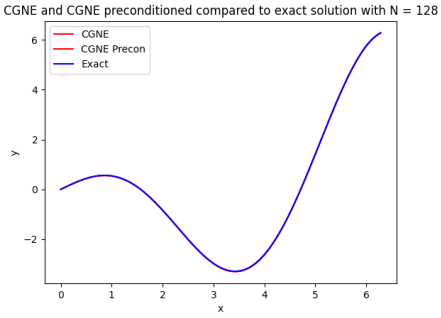
```
Example 4: Rectangular 1D Poisson equation
N=32
CGNE: Solution found after 54 iterations
CGNE Precon: Solution found after 56 iterations
N=128
```

```
CGNE: Solution found after 542 iterations
CGNE Precon: Solution found after 562 iterations
```

The iterations needed to calculate the solution of the 1D Poisson equation is much larger than for the simple square matrix. We will first see if the computed solution is correct by plotting the computed solution in Figure B.5 next to the exact solution. From the figure we can conclude that the solution that is found is correct.



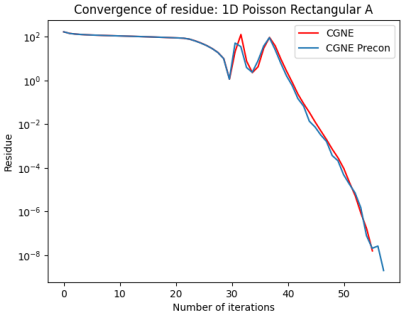
(a) *CGNE and preconditioned solution for N = 32*



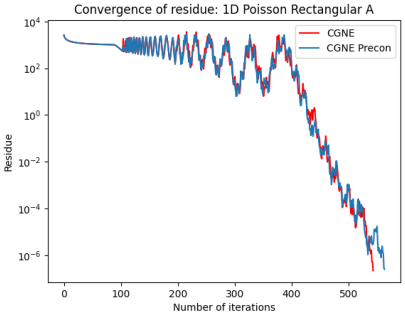
(b) *CGNE and preconditioned solution for N = 128*

Figure B.5 *CGNE and preconditioned solution* $\mathbf{u}$ *compared to the exact solution for* $N = 32$ *and* $N = 128$*. A higher number of grid nodes has a higher accuracy.*

To investigate the high number of iterations needed we will take a look at the residue of each iteration in Figure B.6. After the first n iterations, where n is the number of rows of A, we can see that the residue becomes unstable. This behaviour is due to the the rounding errors reaching 10^{-16}, the smallest machine-representable number.



(a) *CG & CGNE: Residue for Rectangular 1D Poisson with $N = 32$*



(b) *CG & CGNE: Residue for Rectangular 1D Poisson with $N = 128$*

Figure B.6 *Residue for the rectangular 1D Poisson equation for $N = 32$ and $N = 128$ on a logarithmic scale. Both algorithms have a difficult time converging and after n iterations.*

B.3. LSQR: 1D Poisson Equation

In this section we will cover the same 1D Poisson equation and solve them with the LSQR algorithm from [9]. The 1D Poisson equation is

$$-\frac{\mathrm{d}^2 u}{\mathrm{d}x^2} = f \tag{B.15}$$

where we take as an example

$$\mathbf{f} = x\cos(x). \tag{B.16}$$

LSQR will solve the linear system (given in Equation B.17), associated with the central discretization, adding extra columns containing zeros to A^h.

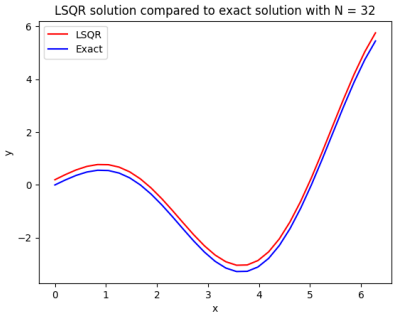
$$\begin{pmatrix} h^2 & 0 & 0 & \dots & \dots & \dots & 0 & 0 & \dots & 0 \\ 0 & 2 & -1 & 0 & \dots & \dots & 0 & \vdots & \ddots & \vdots \\ \vdots & -1 & 2 & -1 & 0 & \dots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \dots & 0 & -1 & 2 & -1 & 0 & \vdots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & -1 & 2 & 0 & \vdots & \ddots & \vdots \\ 0 & \dots & \dots & \dots & 0 & 0 & h^2 & 0 & \dots & 0 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{3N+2} \\ u_{3(N+1)} \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \\ f_{N+1} \end{pmatrix} \tag{B.17}$$

Computing the solution

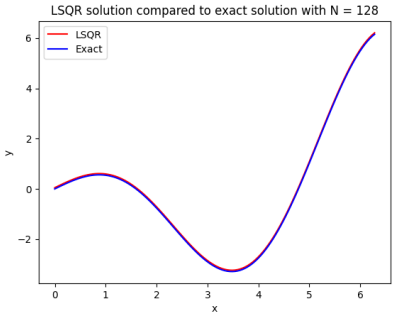
We compute the solution to the 1D Poisson equation with LSQR for both $N = 32$ and $N = 128$.

```
Example 5: LSQR Rectangular 1D Poisson equation
N=32
LSQR: Solution found after 50 iterations
N=128
LSQR: Solution found after 482 iterations
```

The number of iterations needed to find the solution with LSQR is less than that of CGNE. But it is still not the expected n iterations. First we compare the LSQR solution with the exact solution in Figure B.7. Showing us that the LSQR solution is close to the exact solution in fewer iterations.



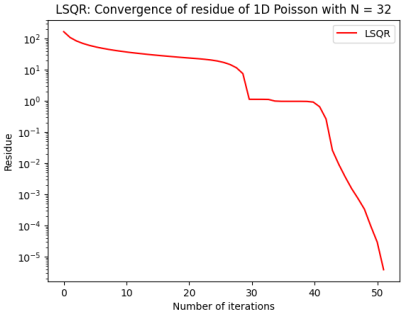
(a) *Residue for Rectangular 1D Poisson with N=32*



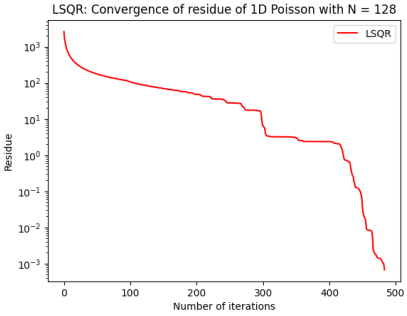
(b) *Residue for Rectangular 1D Poisson with N=128*

Figure B.7 *LSQR solution compared to the exact solution for both N=32 and N=128.*

To investigate why the number of iterations is not equal to n we will plot the residuals on each iteration in Figure B.8. As we can see the residue after iteration n will monotonically decrease. A possible reason for the slow convergence is because of the rounding error being 10^{-16}.



(a) *Residue for Rectangular 1D Poisson with* $N = 32$



(b) *Residue for Rectangular 1D Poisson with* $N = 128$

Figure B.8 *Residue for the rectangular 1D Poisson equation for* $N = 32$ *and* $N = 128$ *on a logarithmic scale.*