Delft University Of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science

Master Thesis
Literature Study Report

Author:
Konrad Kaliszka 1531506

Supervisors:
C.Vuik
M.B. Van Gijzen

TU Delft
Delft University of Technology

Contents

Chapter 1

Description of the problem

1.1 Introduction

Plaxis B.V. is a company specialized in finite element software intended for 2D and 3D analysis of deformation, stability and groundwater flow in geotechnical engineering. Geotechnical applications require advanced constitutive models for the simulation of the non-linear and time-dependent behavior of soils. In addition, since soil is a multi-phase material, special procedures are required to deal with hydrostatic and non-hydrostatic pore pressures in the soil.

Within the finite element formulation large linear systems have to be solved. At this moment fast and robust iterative solvers are available for sequential computing. Since more and more present day computers consists of more cores, Plaxis is working on parallelization of these solvers. It is not so easy to parallelize the current solver with the same amount of iterations. In this project other techniques are investigated in order to develop robust and efficient parallel solvers. As a first approach domain decomposition methods have to be studied. After a good DD method has been selected the properties of this method is investigated for the problems originating from geotechnical applications. A combination of this method with a second level preconditioner is the next step in this investigation. Finally, instead of an arbitrary data partitioning, the decomposition of the computational domain should be based on the physical properties of the domain. This decomposition can be used to have special preconditioners for certain subdomains and can also be used to develop a good second level preconditioner.

1.2 Overview of the report

In this report we are going to present the theory which is will be needed to solve the problem. We will also show some numerical experiments which were done during the literature study period for better understanding the underlying aspects of studied materials. In the end we will draw some preliminary conclusions and state the future directions of the research.

We will start this report with a short description of the Plaxis software and the systems which are solved by it. After that, we make a short introduction to the Finite Element Method, which is done in the next chapter. After that, we are going to describe the Conjugate Methods, one of ingredients of the approach chosen to be the solution for our problem. This chapter is followed by a description of the Deflation method and it's combination with PCG. After that we present the idea of the Domain Decomposition approach, through presentation of two basic methods from this block of mathematics.

In the last chapter, we are going to show results of tests done on the data provided by Plaxis. Also this will be the place of some conclusions and setting goals for the next period.

1.3 Short description of problem

As mentioned before, Plaxis B.V. is a company specialized in finite element software intended for 2D and 3D analysis of deformation, stability and groundwater flow in geotechnical engineering. The soil can consist of different layers and each can have a different material model and/or different parameters within it specification. In addition, there can be specified several structural elements, like sheet pile wall or anchors which may be taken into account when simulating. When a problem is created it is subdivided into a large number of finite elements and equilibrium is solved in several steps and iterations. It is also worth to mention, that Each of these nodes has the ability to move in the x,y and z directions. However some nodes are fixed on the boundary and have limited to no freedom of movement. We use the term *degree of freedom* as the unknowns in the linear system and they will correspond to the direction of nodes in which they are able to move. For each direction one row will be introduced in the matrix, so two or three adjacent rows can correspond to the same node.

An important part of the calculation time, especially for larger 3D projects, is solving a linear system of equations. For large systems, direct solution techniques cannot be used so iterative solution techniques are often used.
We will now present some basic equations and theory, which is used within the Plaxis software. More information can be found in [5].

1.3.1 Deformation Theory

The formula for the static equilibrium can be written in the following way:

$$L^T\sigma + b = 0 \tag{1.1}$$

where $\sigma = [\sigma_x\ \sigma_y\ \sigma_z\ \sigma_{xy}\ \sigma_{yz}\ \sigma_{zx}]^T$ is the stress vector, b is the body forces vector and L^T is the transpose of a differential operator defined as:

$$L^T = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial z} \\ 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial x} & \frac{\partial}{\partial z} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \tag{1.2}$$

We will also use the the kinematic relation, which can be formulated as:

$$\epsilon = Lu \tag{1.3}$$

where ϵ is the stress vector. The link between Eq. (1.1) and (1.3) is formed by a constitutive relation representing the material behaviour i.e., the relation between the rates of stress and strain, which can be written as:

$$\dot{\sigma} = M\dot{\epsilon}, \tag{1.4}$$

and where M is the matrix which represents the relations between rates of stress and strain.

The combination of Eqs. (1.1), (1.3) and (1.4) would lead to a second-order partial differential equation in the displacements u. However, instead of a direct combination, the equilibrium equation is reformulated in a weak form according to Galerkin's variation principle:

$$\int \delta u^T \left(L^T \sigma + b \right) dV = 0 \tag{1.5}$$

In this formulation δu represents a kinematically admissible variation of displacements. Applying Green's theorem for partial integration to the first term in Eq (1.5), we get:

$$\int \delta \epsilon^T \sigma dV = \int \sigma u^T b dV + \int \delta u^T t dS \tag{1.6}$$

This introduces a boundary integral in which the boundary traction appears. The three components of the boundary traction are assembled in the vector t. Eq. (1.6) is referred to as the virtual work equation.

The development of the stress state σ, can be seen as an incremental process:

$$\begin{aligned} \sigma^i &= \sigma^{i-1} + \Delta\sigma \\ \Delta\sigma &= \int \dot{\sigma} dt \end{aligned} \tag{1.7}$$

In this relation σ^i represents the actual state of stress which is unknown and σ^{i-1} represents the previous state of stress which is known. The stress increment $\Delta\sigma$ is the stress rate integrated over a small time increment.
If Eq. (1.6) is considered for the actual state i, the unknown stresses σ^i can be eliminated using (1.7):

$$\int \delta \epsilon^T \Delta\sigma dV = \int \delta u^T b^i dV + \int \delta u^T t^i dS - \int \delta \epsilon^T \sigma^{i-1} dV \tag{1.8}$$

The Eq. (1.8) is then solved with the use of Finite Element Method, which will be described further in this report and that is why we will omit the derivation associated with it. Nevertheless, as the result we end up with the following equation:

$$\int B \delta dV = \int N b^i dV + \int N^T t^i dS - \int B \sigma^{i-1} dV \tag{1.9}$$

,where B is the strain interpolation matrix, which contains the spatial derivatives and of the interpolation functions, and N is a matrix which stores interpolation functions of the displacement vector, known also as the shape functions.

The above equation is the elaborated equilibrium condition in discretised form. The first term on the right-hand side together with the second term represent the current external force vector and the last term represents the internal reaction vector from the previous step. A difference between the external force vector and the internal reaction vector should be balanced by a stress increment $\Delta\sigma$.

The relation between stress increments and strain increments is usually non-linear. As a result, strain increments can generally not be calculated directly, and global iterative procedures are required to satisfy the equilibrium condition (2.13) for all material points.

1.3.2 Global iterative solution procedure

The formula for the global iterative solution procedure has the following form:

$$K^i \Delta v^i = f_{ex}^i - f_{in}^{i-1} \tag{1.10}$$

where K is the stiffness matrix, Δv is the incremental displacement vector, f_{ex} is the external force vector and f_{in} is the internal reaction vector. The superscript i refers to the step number. However, because the relation between stress increments and strain increments is generally non-linear, the stiffness matrix cannot be formulated exactly beforehand. That is why the global iteration process can be written as:

$$K^j \delta v^j = f_{ex}^i - f_{in}^{j-1} \tag{1.11}$$

where the superscript j refers to the iteration number, δv is a vector containing subincremental displacements, which contribute to the displacement increment of step i:

$$\Delta v^i = \sum_{j=1}^{n} \delta v^j \tag{1.12}$$

where n is the number of iterations within step i. The stiffness matrix K, represents the material behavior in an approximated manner. The more accurate the stiffness matrix is, the fever iterations are required to obtain equilibrium within a certain tolerance.

In a its simplest form, K represents a linear-elastic response. In this case, the stiffness matrix can be formulated as:

$$K = \int B^T D^e B dW \tag{1.13}$$

where D^e is the elastic material matrix according to Hooke's law and B is the strain interpolation matrix.

Chapter 2

Finite Element Method

2.1 Introduction

When dealing with partial differential equations, we need to have a tool which will allow us to solve them or at least to get an approximation of the solution. In this chapter we will present a famous method called Finite Element Method, which let us create an approximation of the PDE we are dealing with, which preserves complex geometries and is quite easy to understand. FEM method is used all over the world in thousands of application.

We will illustrate the Finite Element method with the solution of a Poisson equation with Dirichlet boundary condition, where Ω is a bounded open domain in $\mathbb{R}^2$ and Γ is its boundary.

$$-\Delta u = f \tag{2.1}$$

2.2 Variational Equation

To solve this problem approximately, we will need to extract a system of algebraic equations which will yield the solution. To do that, we will use a common approach, namely the weak formulation of the problem. Denote:

$$a(u,u) = \int_\Omega < \nabla u | \nabla u > dx$$
$$(f,v) = \int_\Omega fv \; dx$$

It is easy to show, that a is bilinear. Now, from Green's formula we get:

$$a(u,v) = -(\Delta u, v) = (\nabla u, \nabla v)$$

Hence, now we can reformulate the problem into following one

$$\text{Find } u \in V \quad \text{such that} \quad \forall v \in V: \; a(u,v) = (f,v) \tag{2.2}$$

where $V \subset L_2$ is the subspace of all functions whose derivatives up to first order are in L_2 and which have zeros on Γ. The resulting space is called $H_0^1(\Omega)$. The above condition is called a Variational Equation.

2.3 Galerkin Equations

Let Ω_h denote an approximation of the domain Ω by the union of the m triangles K_i, which come from the triangulation of Ω. Now, we can replace the space V with a finite dimensional space V_h, which is defined as the space of all functions which are piecewise linear and continuous on the polygonal region Ω_h, and which vanish on the boundary Γ. To be more precise:

$$V_h = \{\phi : \phi_{|\Omega_h} \text{ - continuous, } \phi_{I\Gamma_h} = 0, \ \phi_{|K_j} \text{ - linear for all } j\} \tag{2.3}$$

If x_j, where $j \in \{1, ...,n\}$ are the nodes of the triangulation, then a function ϕ_j in V_h, can be actually associated with each of them, so that it satisfies the following condition:

$$\phi_j(x_i) = \begin{cases} 1, \text{if } x_i = x_j \\ 0, \text{if } x_i \neq x_j \end{cases} . \tag{2.4}$$

The above condition makes $\phi_i, i = 1, ..., n$ defined uniquely. Also the ϕ_i's form a basis of the space V_h, so each function now can be represented as a linear combination of them:

$$\forall u \in V_h : \quad u(x) = \sum_{i=1}^{n} \xi_i \phi_i(x) \tag{2.5}$$

If we now recall the variational equation, and write it for the V_h space, then we will get:

$$\text{Find } u \in V_h \quad \text{such that} \quad \forall v \in V_h : \ a(u, v) = (f, v) \tag{2.6}$$

by the linearity of a with respect to v, one can impose the condition $a(u, \phi_i) = (f, \phi_i)$, for $i = 1, ..., n$. But from (2.5), we know that u can be represented as the linear combination of the basis function. If we combine those two facts, we will get:

$$\sum_{i=1}^{n} \alpha_{ij} \xi_j = \beta_j, \text{ for all } i = 1, ...n. \tag{2.7}$$

where $\alpha_{ij} = a(\phi_i, \phi_j)$ and $\beta_i = (f, \phi_i)$.

The above equation allows us to formulate a linear system:

$$Ax = b \tag{2.8}$$

with $A = [\alpha_{ij}]_{n \times n}$ and $b = [\beta_1 \ ... \ \beta_n]^T$

The matrices generated by this method have some nice properties. The most important are the facts, that A is Symmetric Positive Definite and sparse. Knowing this, we may now use one of the CG variants for solving these linear systems.

Chapter 3

Conjugate Gradient Method

3.1 Introduction

One of our main problems is to solve a linear system:

$$Ax = b, \tag{3.1}$$

where $A \in \mathbb{R}^{n \times n}$ is called a coefficient matrix, $b \in \mathbb{R}^n$ a right-hand side vector, and where $n \in \mathbb{N}$. Keeping in mind the fact, that we are dealing with matrices from the discretization, we will assume that A is symmetric and semi-positive definite.

There are many ways to solve this problem. In this chapter we will focus on a well-known iterative method, called Conjugate Gradient or, in short-hand notation CG, developed by E. Stiefel and by M.R Hestenes[**3**], which allows us to compute the solution of the above mentioned system of equations. The success of this algorithm lies in it's simplicity. However, to describe the Conjugate Gradient method precisely, we have to know exactly what is a basic iterative method.

3.2 Basic Iterative Methods

Basic iterative solution methods are used to generate a sequence $(x_i, i = 0, 1, ...)$ which may be finite or not, consisting of the approximations of the exact solution x. The compute this sequence, the following recursive formula is used,

$$x_{i+1} = x_i + M^{-1}(b - Ax_i) \tag{3.2}$$

We can substitute $r_i = b - Ax_i$, to which we will from now on refer as the i-th residual, which is used to measure the difference of the i-th approximation and the exact solution and rewrite the above equation once again in a more pleasant way,

$$x_{i+1} = x_i + M^{-1}r_i \tag{3.3}$$

If we now write the first steps of this iteration process,

$$
\begin{aligned}
&x_0 = x_0, \\
&x_1 = x_0 + M^{-1}r_0, \\
&x_2 = x_1 + M^{-1}r_1 = x_0 + M^{-1}r_0 + M^{-1}(b - Ax_0 - AM^{-1}r_0) \\
&= x_0 + 2M^{-1}r_0 - M^{-1}AM^{-1}r_0 \\
&\quad \vdots
\end{aligned} \tag{3.4}
$$

we can conclude that

$$x_i \in x_0 + \text{span}\{M^{-1}r_0, M^{-1}A(M^{-1}r_0), ..., (M^{-1}A)^{i-1}(M^{-1}r_0)\} \tag{3.5}$$

The subspace which occurs in the last formula is actually a special case of a Krylov-space, which is defined as $K^i(A, r_0) = \text{span}\{r_0, Ar_0, ..., A^{i-1}r_0\}$. From this we conclude that for each basic iterative method the following is fulfilled

$$x_i \in x_0 + K^i(M^{-1}A; M^{-1}Ar_0) \tag{3.6}$$

These methods are also called Krylov(-subspace) methods. We see that there are two problems which arise from the formula of the basic iterative method. Given the matrix M^{-1}(which also called a "preconditioner") the first problem is to find a suitable basis for $K^i(:,:)$ such that the iterative method has a fast convergence rate with a reasonable accuracy and efficiency with respect to memory storage and computational time. Second, is actually finding the x_i.

3.3 Conjugate Gradient Method

The present section will be devoted to a description of the Conjugate Gradient Method, which nowadays is probably the best known and mostly used iterative method for solving SPD linear systems.

To explain how the CG method works, let us define first what an A-inner product and what an A-norm is.

Definition 3.1. *The A-inner product is defined by*

$$< x|y >_A = x^T Ay$$

Definition 3.2. *The A-semi-norm is defined by*

$$||x||_A = \sqrt{< x|x >_A}$$

Whenever A is Positive Define, we may talk of an A-norm.

The underlying idea of CG is very simple. The sequence $(x_j, j = 0, 1, 2, ...)$ should have the following property:

$$||x - x_j||_A = \min_{y \in K^j(A;r_0)} ||x - y||_A, \text{ for all } j. \tag{3.7}$$

We are sure about the existence of the minimum only if A is SPD. However due to our knowledge about the matrices coming from the discretization of PDE's we do not have to worry about this.

Notice that

$$||x - x_1||_A^2 = (x - \alpha_0 r_0)^T A(x - \alpha_0 r_0) = x^T Ax - 2\alpha_0 r_0^T Ax + \alpha_0^2 r_0^T Ar_0 \tag{3.8}$$

Which can be considered as a parabola of the variable α_0. Hence the minimum is achieved for $\alpha_0 = \frac{r_0^T Ax}{r_0^T Ar_0} = \frac{r_0^T b}{r_0^T Ar_0}$.

In the steepest descent method, each next iteration step is determined by the formula

$$x_{k+1} = x_k + \alpha_k p_k, \tag{3.9}$$

where p_k is the direction of minimum search function for the energy of the system. If we multiply the above equation by A and subtract b from it, we will get

$$Ax_{k+1} - b = Ax_k - b + \alpha_k Ap_k$$

But this nothing else but

$$r_{k+1} = r_k - \alpha_k Ap_k. \tag{3.10}$$

If we now assume, that p and p_k are conjugate, then from

$$< p|r_{k+1} >=< p, |r_k > +\alpha_k < p, Ap_k > \tag{3.11}$$

we see, that if $< p|r_k >= 0$, then also $< p|r_{k+1} >= 0$. This the main condition of the CG method, that for each $j = 0, 1, ..., k$ we have that $< p_j|r_{k+1} >= 0$. If we now define

$$p_0 = -r_0, \ \frac{r_0^T Ax}{r_0^T Ar_0} = \frac{< r_k|p_k >}{< p_k|p_k >} \text{ and } p_{k+1} = -r_k + \beta_k p_k \text{ for } k = 0, 1, 2, ... \tag{3.12}$$

where β_k induce that p_{k+1} and p_k will be conjugate, then we are done. The only thing which is left to do, is to find β_k. To do that, let us notice that

$$< p_{k+1}|p_k >_A =< -r_{k+1} + \beta_k p_k|p_k >_A = - < r_{k+1}|p_k >_A + \beta_k < p_k|p_k >_A = 0$$

$$\text{when } \beta_k = \frac{<r_{k+1}|p_k>}{<p_k|p_k>_A}.$$

Knowing all those facts, we can write a pseudo-code of the Conjugate Gradient algorithm. However, in most of the literature, for example [**9**], the coefficients α_k and β_k, are computed in a slightly different way. The pseudo-code which is presented is using the ones which are used more often. Later we will refer to the new coefficients as "Official Approach" and "Proof Approach" to the ones which where defined during the derivation of the CG Method.

Conjugate Gradient Algorithm

Choose x_0, set $i = 0, r_0 = b - Ax_0$.
WHILE $r_k \not= 0$ **DO**
$i := i + 1$
IF $i = 0$ **DO**
$p_1 = r_0$
ELSE
$\beta_i = \frac{r_{i-1}^T r_{i-1}}{r_{i-2}^T r_{i-2}}$
$p_i = r_{i-1} + \beta_i p_{i-1}$
ENDIF
$\alpha_i = \frac{r_{i-1}^T r_{i-1}}{p_i^T A p_i}$
$x_i = x_{i-1} + \alpha_i p_i$
$r_i = r_{i-1} - \alpha_i A p_i$
END WHILE (3.13)

It is very important to notice, that to use CG we need only to remember four vectors and one matrix which makes it attractive in the usage of memory space.

From [**6**], we now that the convergence rate of the CG-method can be easily estimated using the following theorem:

Theorem 3.3. *Let A and x be the coefficient matrix and the solution of (1.1), and let $(x_i, i = 0, 1, 2...)$ be the sequence generated by the CG method. Then, elements of the sequence satisfy the following inequality:*

$$||x - x_i||_A \leq 2 \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^i ||x - x_0||_A, \tag{3.14}$$

where $\kappa(A)$ is the condition number of A in the 2 - norm.

We clearly see, that the convergence depends on the condition number of A, hence we can conclude that the closer $\kappa(A)$ is to 1, the faster we approach the solution of the (3.1). Therefore it is desired to have a matrix with as low as possible condition number. This leads us to a modification of the CG, called Preconditioned Conjugate Gradient Method.

3.4 Preconditioned Conjugate Gradient Method

The idea which helps us to escape the barrier created by Theorem 3.3 and improve the efficiency and robustness of CG is to transform the original linear system (3.1) into one which has the same solution, but is easier to solve with CG.
Let us consider the following problem:

$$A^* x^* = b^*, \tag{3.15}$$

where $A^* = P^{-1}AP^{-T}$, $x^* = P^{-T}x$ and $b^* = P^{-1}b$, where P is a non-singular matrix. The SPD matrix M defined by $M = PP^T$ is called the preconditioner. We can now use the original CG algorithm to solve our new system. The result is the algorithm for the Preconditioned Conjugate Gradient method, or in short-hand notation PCG-method. However, the presented pseudo-code will be rewritten in such a way, that we will only use quantities without the * sign occurs.

Preconditioned Conjugate Gradient Algorithm

Choose x_0, set $i = 0, r_0 = b - Ax_0$.
WHILE $r_i \neq 0$ **DO**
$z_i = M^{-1}r_i$
$i := i + 1$
IF $i = 0$ **DO**
$p_1 = z_0$
ELSE
$\beta_i = \frac{r_{i-1}^T z_{i-1}}{r_{i-2}^T z_{i-2}}$
$p_i = z_{i-1} + \beta_i p_{i-1}$
ENDIF
$\alpha_i = \frac{r_{i-1}^T z_{i-1}}{p_i^T A p_i}$
$x_i = x_{i-1} + \alpha_i p_i$
$r_i = r_{i-1} - \alpha_i A p_i$
END WHILE (3.16)

From Theorem (1.1), which determines the convergence rate, we see, that in PCG, $\kappa(P^{-1}AP^{-t})$ will be the coefficient telling us about the speed. That's why the success will be depending on a good choice of the matrix P.

There are two extreme choices, which show the range of PCG. If $P = I$, we will go back to the original CG-method, whereas if we choose $PP^T = A$, we will converge to the solution in one iteration. There are many possibilities of choosing the preconditioner. However, we should keep in mind, that the more complex our preconditioner will be, the more time we will spend on construction and application in the program. That's while in this report we will present only two easy precoditioners, to show the possible choices.

If we, take as M, the diagonal of matrix A, we will be dealing with the most standard preconditioner, called Jacobi-preconditioner, due to the origins in the Jacobi-method. The reason to choose this matrix is the fact, that it is easy to construct, the matrix multiplication is very fast, because of the big number of zero elements. At last, but not least $diag(A^*) = 1$, which results in saving n multiplications in the matrix vector product.

The other proposition for the matrix M is to take a precondtioner of the following form:

$$M = \frac{1}{2-\omega}(\frac{1}{\omega}D + L)(\frac{1}{\omega}D)^{-1}(\frac{1}{\omega}D + L)^T \tag{3.17}$$

where D and L are the diagonal and the ower triangular of A respectively.

This preconditioner is called SSOR, and its name due to the connection with SSOR-method. The optimal value of the parameter ω, like the parameter in the SOR method, will reduce the number of iterations to a lower order. Although in practice, the spectral information needed to get the optimal ω is quite expensive in the computational sense.

3.5 Numerical illustration

In this section, we will present some numerical experiments, which were done to show how CG and PCG work in practice. For that, we chose a SPD matrix, on which we will perform methods presented in this chapter, namely we will deal here with the CG with "proof coefficients", CG with "Official coefficients", PCG with Jacobi preconditioner "Proof coefficients", PCG with Jacobi preconditioner and "Official coefficients", PCG with SSOR preconditioner "Proof coefficients", PCG with SSOR preconditioner and "Official coefficients", PCG with Jacobi and SSOR preconditioner and "Official coefficients". For SSOR we took $\omega = 0.5$ and for SSOR with Jacobi we took $\omega = 1.9$.

Below we can find the matrix which was used for testing.

$$A = \begin{bmatrix} 10 & -4 & 1 & & & & & \\ -4 & 11 & -4 & 1 & & & 0 & \\ 1 & -4 & 11 & -4 & 1 & & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & 1 & -4 & 11 & -4 & 1 \\ & 0 & & & 1 & -4 & 11 & -4 \\ & & & & & 1 & -4 & 10 \end{bmatrix}$$

with $n = 40$, and vector $b = [1\ 2\ 3\ 4\ \ldots\ 40]^T$. On the next page we can see the plot of the iteration step versus the log of error.

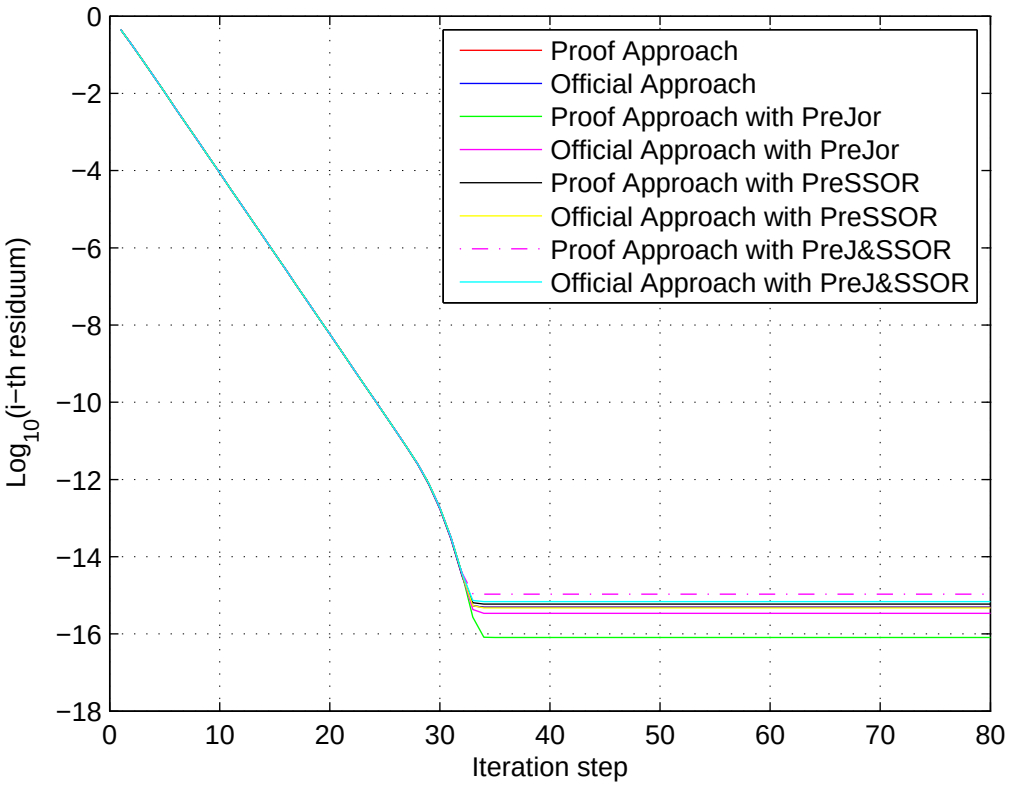

Figure 3.1: Plot of $log_{10}(i\text{-th residuum})$ for system Ax=b

From the presented data, we clearly see that the Conjugate Gradient Methods works perfectly for the given system, which was expected, because matrix A a symmetric strongly diagonally dominant which implies that is SPD and which allows us to expect convergence.
The reason for the fast convergence of all methods is the conditional number of matrix A, which is quite low. To be precise, it is equal to 4.988502495710613.

However, if we now change the system, and instead of the presented one, we will test these CG variants for the system, which we get from a discreatization of a problem created in Plaxis software, namely Second Simple Problem (Which is described in the last chapter of this report), we will get a different behavior of the iteration processes. On the next page we will present the figure which will show us the convergence of the CG methods.

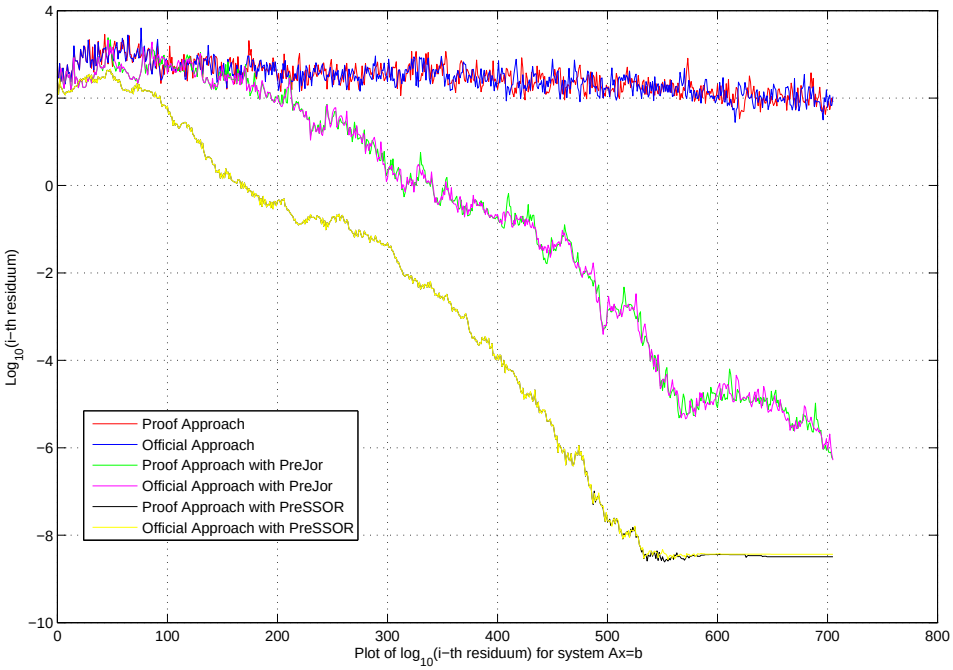

Figure 3.2: Plot of $log_{10}(i\text{-th residuum})$ for Second Simple Problem

We see, that in this case we can notice a big difference between the methods. For example, normal CG does not even get close to the solution during the whole iteration.
If we now look at the conditional number of the system, we will see the reason for those results, because $cond(A) = 3.042123456938706 * 10^8$, while $cond(A') = 1.122208310761762 * 10^6$, where A' is the preconditioned coefficient matrix by PreSSOR. This example shows us, that the use of Preconditioned Conjugate Gradient method is a right choice for nontrivial linear systems.

Chapter 4

Deflation

4.1 Introduction

In the previous chapter we consider the CG-method for solving linear systems with a matrix A which is SPD. We also present a way to improve it's convergence rate which depends mainly on the condition number of the coefficient matrix, by introducing a transformation of the original linear system into a new one with a smaller condition number. This approach was called preconditioning. We also presented some of the classical preconditioners. In this chapter we will show another way of preconditioning called deflation.

4.2 Deflation

Definition 4.1. *Let A be an SPD coefficient matrix from (1.1). Suppose that $Z \in \mathbb{R}^{n\times k}$, with full rank, and $k \geq n-d$ is given and d is the number of zero eigenvalues of A. Then the deflation matrix $P \in \mathbb{R}^{n\times n}$ is defined as follows:*

$$P := I - AQ \tag{4.1}$$

where:

- $Q := ZE^{-1}Z^T$ *is called the correction matrix.*
- $E := Z^TAZ$ *is called Galerkin matrix.*

Z is often called "deflation-subspace matrix" whose k columns are the "deflation vectors" or "projection vectors". Right now, they do not need to be specified. However, they will be chosen in such a way, that matrix E will be nonsingular[9].

We will now go back to our original linear problem, and solve it using the following decomposition of the solution vector.

$$x = (I - P^T)x + P^Tx \tag{4.2}$$

notice, that

$$I - P = I - (I - AQ) = AQ \tag{4.3}$$

$$AP^T = A(I - AQ)^T = A(I - QA) = A - AQA = (I - AQ)A = PA \tag{4.4}$$

$$E^T = (Z^T A Z)^T = Z^T A^T Z = Z^T A Z = E \tag{4.5}$$
$$Q^T = (Z E^{-1} Z^T)^T = Z E^{-T} Z^T = Z E^{-1} Z^T = Q \tag{4.6}$$

Let us now go back to (2.2)

$$\begin{aligned} x &= (I - P^T)x + P^T x \\ x &= Qb + P^T x \\ Ax &= AQb + AP^T x \\ b &= AQb + PAx \\ (I - AQ)b &= PAx \\ Pb &= PAx \end{aligned} \tag{4.7}$$

It is crucial to notice, that the solution of (4.7) does not have to be a solution of the original linear system(1.1), because PA is singular. That's why, we will denote the solution of (4.7) as $\bar{x}$ to distinguish from x. We may now formulate a deflated system of our original problem as:

$$PA\bar{x} = Pb, \tag{4.8}$$

and solve it using CG. However we need still to connect the solutions of (3.1) and (4.7), otherwise the whole procedure would not have any reason to exist. The following Lemma [**9**]will provide the needed link:

Lemma 4.2. *Let P be the deflation matrix and Q be the correction matrix of the (1.1) under the assumption that Z satisfies the requirements of Definition (2.1) and b is the right hand-side of (1.1). Suppose that x be the solution of (3.1) and $\bar{x}$ be the solution of (4.8). Then, the following formula holds*

$$x = Qb + P^T \bar{x} \tag{4.9}$$

Proof. Notice that if we decompose $\bar{x}$ as

$$\bar{x} = x + y,$$

where $y \in \mathcal{R}(Z) \subset \mathcal{N}(PA)$, then

$$P^T \bar{x} = P^T x + P^T y = P^T x, \tag{4.10}$$

because $P^t y = \mathcal{O}_n$. This property have arisen from the fact that

$$P^T Z = (I - QA)Z = Z - QAZ = Z - Z = \mathcal{O}_{n \times k} \tag{4.11}$$

Hence, now it is easy to see that:

$$x = (1 - P)^T x + P^T x = Qb + P^T \bar{x}.$$

□

It can be shown, that PA is SPSD, hence it can be interpreted as the new coefficient matrix of the linear system (4.8).

4.3 Deflated CG and PCG Methods

We can now write the pseudo-code of the deflated CG method:

Deflated Conjugate Gradient Algorithm

Choose $\bar{x}_0$, set $i = 0, \bar{r_0} = P(b - A\bar{x}_0)$.
WHILE $\bar{r_k} \not= 0$ **DO**
$i := i + 1$
IF $i = 0$ **DO**
$p_1 = \bar{r}_0$
ELSE
$\beta_i = \dfrac{\bar{r}_{i-1}^T \bar{r}_{i-1}}{\bar{r}_{i-2}^T \bar{r}_{i-2}}$
$p_i = \bar{r}_{i-1} + \beta_i p_{i-1}$
ENDIF
$\alpha_i = \dfrac{\bar{r}_{i-1}^T \bar{r}_{i-1}}{p_i^T PAp_i}$
$\bar{x}_i = \bar{x}_{i-1} + \alpha_i p_i$
$\bar{r}_i = \bar{r}_{i-1} - \alpha_i PAp_i$
END WHILE
$x_{orginal} = Qb + P^T \bar{x}_{last}$ (4.12)

We see that the algorithm is barely touched, the are only little differences between it and the original CG algorithm. We can also make a preconditioning of the system by using an SPD precondtioner M^{-1},and then apply onto it Deflated CG method. As the result we get Deflated Preconditioned CG Method, for which present the pseudo-code on the next page.

Deflated Preconditioned Conjugate Gradient Algorithm

Choose $\bar{x}_0$, set $i = 0, \bar{r_0} = P(b - A\bar{x}_0)$.
WHILE $\bar{r_k} \not= 0$ **DO**
$i := i + 1$
IF $i = 1$ **DO**
$y_0 = M^{-1}\bar{r}_0$
$p_1 = y_0$
ELSE
$y_{i-1} = M^{-1}\bar{r}_{i-1}$
$\beta_i = \frac{\bar{r}_{i-1}^T y_{i-1}}{\bar{r}_{i-2}^T y_{i-2}}$
$p_i = y_{i-1} + \beta_i p_{i-1}$
ENDIF
$\alpha_i = \frac{\bar{r}_{i-1}^T \bar{r}_{i-1}}{p_i^T PAp_i}$
$\bar{x}_i = \bar{x}_{i-1} + \alpha_i p_i$
$\bar{r}_i = \bar{r}_{i-1} - \alpha_i PAp_i$
END WHILE
$x_{orginal} = Qb + P^T \bar{x}_{last}$ (4.13)

4.4 Deflation Vectors

The choice of the deflation vectors is a very important part of the whole process of deflation methods. In literature [**9**]we can find several proposition for the candidates to use. The most known strategies for construction of those vectors are:

- Approximated Eigenvector Deflation Vectors
- Recycling Deflation Vectors
- Subdomain Deflation Vectors
- Multigrid and Multilevel Deflation Vectors
- Rigid Body Modes

It is worth to mention, that right know we do not have a universal strategy for constructing the deflation vectors, which gives the best result for every problem. In this chapter we will restrict ourself to present only two strategies, namely Subdomain Deflation and Rigid Body Modes.

4.4.1 Subdomain Deflation Vectors

In this variant of deflation, we choose the deflation vectors in the following way: Let $q > 1$ and $j \in \{1\ ,\ .\ .\ .\ ,\ q\}$. We divide the computational domain Ω into q

subdomains Ω_j, by the following rules:

$$\bar{\Omega} = \cup_{j=1}^{q} \bar{\Omega}_j \ \wedge \ \forall_{i \neq j} \ \Omega_i \cap \Omega_j = \emptyset \tag{4.14}$$

Let's also denote Ω_h and Ω_{h_j}, for the discretized domain and subdomains respectively. After that we can introduce the deflation vector z_j associated with the j-th subdomain as follows:

$$z_j(i) = \begin{cases} 0, \ x_i \in \Omega_h \backslash \Omega_{h_j} \\ 1, \ x_i \in \Omega_{h_j} \end{cases} . \tag{4.15}$$

After this step, we define $Z = [z_1 \ z_2 \ \ . \ . \ . \ z_q]$. This finish the construction.

This method is strongly related to approaches known as Domain Decomposition Methods.

4.4.2 Rigid Body Modes

In the recent research in the field of deflation, we can find another approach for choosing the deflation vectors. In [13], we may find an introduction to the Rigid Body Modes used as the engine for the deflation vectors. The main idea is to set for the i-th deflation vector the i-th vector of the null space of A_s, which is a submatrix created from the elements from the FEM discreatisation, which are composing the aggregate subdomains.

Chapter 5

Domain Decomposition Methods

5.1 Introduction

With the rapid growth of high speed computing, we get a powerful tool to our hand. Multi-core processors gives us a possibility to solve very big computation problems in a much faster way than the traditional sequential ones, using the advantages which come from the architecture of the machine used to compute. Among techniques which are based on the parallelization of the computation process, domain decomposition methods are undoubtedly the best known and perhaps the most promising for the problem studied by Plaxis. These methods combine ideas from Partial Differential Equations, linear algebra, mathematical analysis and some part of graph theory. In this chapter we will focus on the decomposition methods, which are based on the general concepts of graph partitioning.

Definition 5.1. *We will call a method a Domain Decomposition method, if its main idea will be based on the principle of divide and conquer applied on the domain of the problem.*

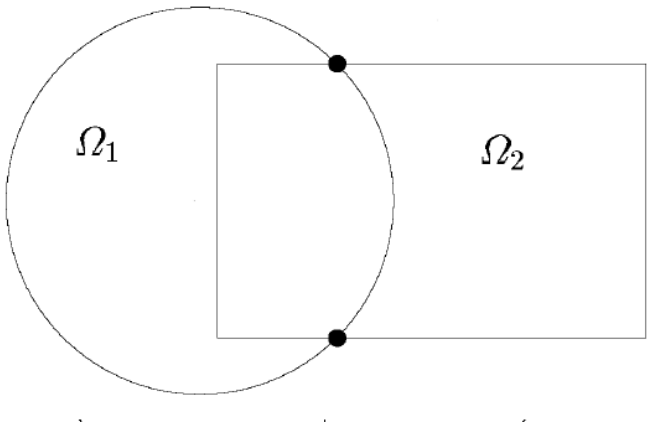

Figure 5.1: An example of domain decomposition

Let us consider the following problem. We want to solve the Laplace Equation on domain Ω partitioned as shown in the figure above. Domain Decomposition methods attempt to solve the problem on the entire domain

$$\Omega = \bigcup_{i=1}^{s} \Omega_i \tag{5.1}$$

from the problem solution on the subdomain Ω_i. There are several reasons why this approach can be advantageous. First of all, the subdomains may have a simpler geometry then Ω. Also sometimes the problem may have a natural split into smaller regions, in which we can have different equations that describe the model. However maybe the most important reason to use Domain Decomposition Methods is the fact, that they are the best choice for the solution of a problem, if we want to parallelize the computational process. Last, but not least, they allow us to deal with the lack of memory, by splitting the domain into parts which will fit into our computers.

There are several methods in the Domain Decomposition family. This report presents only some of them.

5.2 Schwarz Alternating Procedures

The earliest known domain decomposition method is the alternating method of H. Schwarz dating back to 1870. It consisted of three parts: alternating between two overlapping domains, solving the Dirichlet problem on one domain at each iteration and taking boundary conditions based on the most recent solution obtained from the other domain.
Let's consider a domain Ω as shown in Figure 5.1 with two overlapping subdomains Ω_1 and Ω_2 on which we want to solve a PDE of the following form:

$$\begin{cases} Lu = f, & \text{in } \Omega \\ u = g, & \text{on } \partial\Omega \end{cases} . \tag{5.2}$$

Let $\partial\Omega$ denote the boundary of Ω and the artificial boundaries, Γ_i, are the part of the boundary of Ω_i that is interior to Ω, and s is the number of subdomains. Schwarz Alternating Procedure (SAP) for s subdomain problem will be of the following form:

Schwarz Alternating Procedure

Choose u_0
WHILE no convergence **DO**
FOR $i = 1, ...s$ **DO**
Solve $Lu = f$ in Ω_i with $u = u_{ij}$ in Γ_{ij}
Update u values on Γ_{ij}, $\forall j$
END FOR
END WHILE (5.3)

In our case, $s = 2$.

In many applications, it is possible to use a matching grid in the overlap region to avoid the duplication of the unknowns on the overlap. The matching version of the alternating method is known as the **multiplicative Schwarz method** (MSM). Writing the linear system for the discretized problem as $Au = f$, we can write the iteration in two fractional steps:

$$\begin{aligned} u^{n+1/2} &= u^n + \begin{bmatrix} A_{\Omega_1}^{-1} & 0 \\ 0 & 0 \end{bmatrix} (f - Au^n) \\ u^{n+1} &= u^{n+1/2} + \begin{bmatrix} 0 & 0 \\ 0 & A_{\Omega_2}^{-1} \end{bmatrix} (f - Au^{n+1/2}) \end{aligned} \tag{5.4}$$

where A_{Ω_i} stays for the discrete form of the operator L restricted to Ω_i.
We can easily see, that the main part of the multiplicative Schwarz method is sequential, so it cannot directly use the benefits of the multi-core architecture and therefore it is not a suitable choice when making a parallel solver.

In literature **[10]** we can find also another approach to SAP, which is more parallel-oriented. This method, called **Additive Schwarz method** (ASM), can be considered as a parallel version of the multiplicative Schwarz method. Its main idea is to change presented previous algorithm by combining the computation of influences to the solution from of each subdomain into one iteration, instead of doing this in each step. For our our example with two domain, the iteration step can be written as:

$$u^{n+1} = u^n + \left(\begin{bmatrix} A_{\Omega_1}^{-1} & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & A_{\Omega_2}^{-1} \end{bmatrix} \right) (f - Au^n) \tag{5.5}$$

If we make a substitution of $B_i = R_i^t A_{\Omega_i}^{-1} R_i$, where R_i is the rectangular restriction matrix that returns the vector of components defined in the interior of Ω_i, then the above equation will be of the following form,

$$u^{n+1} = u^n + (B_1 + B_2)(f - Au^n) \tag{5.6}$$

Having this equation, we can easily generalized ASM for s number of subdomains, by simply adding $B_i(f - Au^n)$ to the right-hand side. As the result of this, for a domain $\Omega = \bigcup_{i=1}^{s} \Omega_i$,we can write the algorithm for Additive Schwarz Method in a following form

Additive Schwarz Method

Choose $u_0,\ i = 0,$
WHILE no convergence **DO**
$r_i = b - Au^n$
FOR $i = 1, ...s$ **DO**
$\delta_i = B_i r_i$
END FOR
$u^{n+1} = u_n + \sum_{i=1}^{s} \delta_i$
$i = i + 1$
END WHILE (5.7)

5.3 Schur Complement

Let's consider a following problem:

$$\begin{aligned} Lu &= f \text{ in } \Omega \\ u &= g \text{ on } \partial\Omega \end{aligned} \tag{5.8}$$

with the domain Ω partitioned onto s subdomains. After discretization of the problem, we can label the nodes by subdomain in a specific way, so the linear system will have a following structure:

$$\begin{bmatrix} B_1 & & & & E_1 \\ & B_2 & & & E_2 \\ & & \ddots & & \vdots \\ & & & & E_s \\ F_1 & F_2 & \dots & F_s & C \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_s \\ y \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_s \\ g \end{bmatrix} \tag{5.9}$$

where each x_i represents the subvector of unknowns that are interior to subdomain Ω_i, and y represents the vector of all interface unknowns. It is useful to write the system in a more simple form, i.e.

$$A \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix} \text{ ,where } A = \begin{bmatrix} B & E \\ F & C \end{bmatrix} \tag{5.10}$$

and where E represents the subbdomain to interface coupling seen from the subdomains, while F represents the interface to subdomain coupling seen from the interface nodes. To illustrate this, let us consider a domain split into only two subdomains. Let's assume that the subdomains are of the same size and both are squared. Then an illustrative mesh and corresponding coefficient matrix A will look like

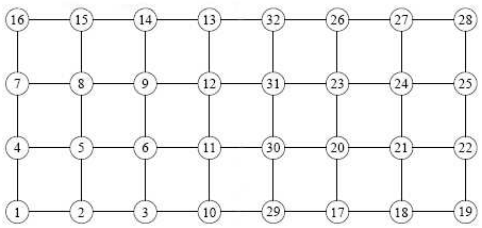

Figure 5.2: An exemplary mesh for the problem described above.

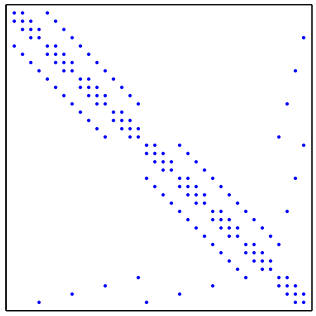

Figure 5.3: Matrix associated with the finite difference mesh of the figure 5.2

Now, we can easily express x with the new terms. From the the first equation it follows that

$$x = B^{-1}(f - Ey) \tag{5.11}$$

If we now substitute this into the second equation, we will obtain a *reduced system*,

$$Sy = g - FB^{-1}f \tag{5.12}$$

where the matrix S is called the *Schur complement* and is created by the following rule:

$$S = C - FB^{-1}E \tag{5.13}$$

If we can form S and solve the associated linear system, then the interface variable y can be obtained. From this we can easily obtain the remaining variable x. Because of the block structure of B, we notice that the solution of the system reduce to solving s separate systems. Because the sets of the variables in each of the system are disjoint, we can solve them simultaneously in parallel. This approach is called *Block Gaussian Elimination* (BGE). The algorithm for it will be now presented:

Block Gaussian Elimination Algorithm

$$\begin{aligned}
&\textbf{SOLVE} \quad BE' = E, \quad Bf' = f \\
&\textbf{COMPUTE} \quad g = g - Ff', \quad S = C - FE' \\
&\textbf{SOLVE} \quad Sy = g' \\
&\textbf{COMPUTE} \quad x = f' - E'y
\end{aligned} \tag{5.14}$$

The partitioning used for the BGE method was edge-based. It means, that a given edge in the graph does not straddle two domains and if any two vertices are coupled, they have to belong to the same subdomain. In the graph theory, this point

of view is less common than the vertex-based partitioning, in which a vertex is not shared by two subdomains (except when subdomains overlap).

We will call interface edges all edges which link vertices that are not in the same subdomain. Interface vertices will be those vertices in a given subdomain, that are adjacent to an interface edge. Now due to the fact, that we split the domain according to a new rule, we change the ordering of the nodes. Now the interface nodes are labeled as the last nodes in each subdomain. To illustrate this, let us recall the example used to present edge-based partitioning and apply new rules to it. As the result we will receive the following mesh and coefficient matrix:

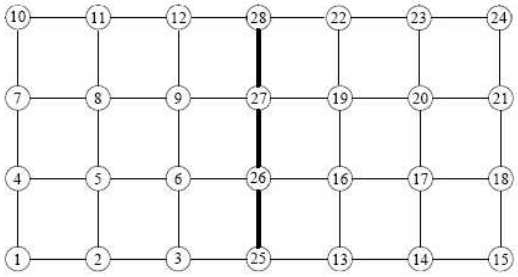

Figure 5.4: An exemplary mesh for the problem described above

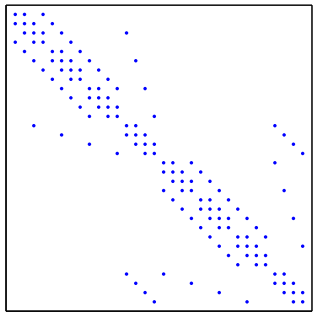

Figure 5.5: Matrix associated with the finite difference mesh of the figure 5.4

Let us consider now the Schur complement system obtained with the new numbering of the nodes. The coefficient matrix A now has a natural s-block structure. For example, if $s = 2$, the matrix will be of the following form:

$$A = \begin{bmatrix} A_1 & A_{12} \\ A_{21} & A_2 \end{bmatrix}. \tag{5.15}$$

Also for each subdomain, the variables will be now have it's own local structure, i.e.,

$$z_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

where x_i now denotes interior nodes, while y_i denotes the interface nodes associated with subdomain i. Now each matrix A_i, will be called local matrix and it will have similar structure to matrix A from (4.9),

$$A_i = \begin{bmatrix} B_1 & E_i \\ F_i & C_i \end{bmatrix}. \tag{5.16}$$

as before, B_i represents the matrix associated with the internal nodes of subdomain i, E_i and F_i represents the susbdomain to interface coupling seen from the subdomains and interface to subdomain coupling seen from the interface nodes respectively and C_i will be the local part of the interface matrix C, which represents the coupling between local interface nodes. Matrices A_{ij} contains zero sub-block in the part that acts on the variable x_j, therefore we can write that

$$A_{ij} = \begin{bmatrix} 0 \\ C_{ij} \end{bmatrix}$$

It is worth to mention, that most of the C_{ij} matrices are zero, since only those indices j of the subdomains that have coupling with subdomain i will yield a nonzero C_{ij}.

If we now write the part of linear system that is local to subdomain i, as

$$\begin{array}{lcl} B_i x_i + E_i y_i & = & f_i \\ F_i x_i + C_i y_i + \sum_{j \in N_i} C_{ij} y_j & = & g_i \end{array}. \tag{5.17}$$

The term $C_{ij}y_j$ is the influence coming from the neighboring subdomain with number j. N_j is a set of indexes of the subdomains which are adjacent to subdomain j. If we assume that B_i are nonsingular, then we can apply the similar solution technique, which we used to develop the BGE. As the result of this, we receive a system of reduced systems

$$S_i y_i + \sum_{j \in N_i} E_i j y_j = g_i - F_i B_i^{-1} E_i \tag{5.18}$$

where S_i is the "local" Schur complement, and is defined as

$$S_i = C_i - F_i B_i^{-1} E_i. \tag{5.19}$$

5.4 Numerical Illustration

The presented numerical experiments in this section illustrate how does the theory from this paragraph acts in practice. Both presented methods, namely Schur Complement Method and Schwarz have been tested on the standard test equation, i.e. on the Poisson equation

$$-\Delta u = f, \tag{5.20}$$

on the domain Ω, which was chosen to be of a rectangular shape, and with Dirichlet boundary condition. However due to the fact, that we are going to work with the Schwarz method in the future, we restrict ourself only to present results only for this method.

The domain Ω after discretization has a $n \times 2n + 1$ size. We decided to split it into two subdomains Ω_1 and Ω_2 along the vertical middle, with a overlap at it. Below we can see a graphical illustration of this process, when $n = 4$.

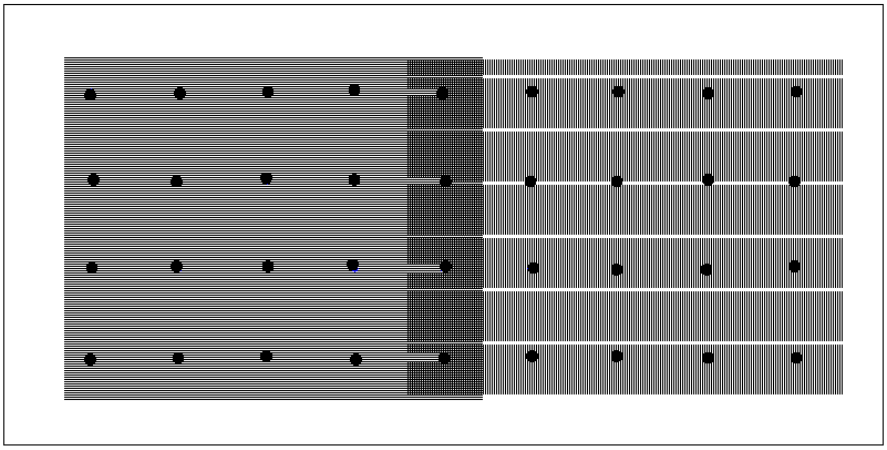

Figure 5.6: Domain Ω split into two subdomains, Ω_1 and Ω_2.

We decided to enumerate the nodes along the columns, i.e. the node at (i, j) position will be consider to be $i + (j - 1)n$-th node in the ordering.

Now we are going to see how does the Schwarz alternating process do works for this problem. We will check both standard variants of SAP, i.e. Schwarz Additive and Schwarz Multiplicative Method. We will not only present only how does the convergence rate for each of the methods looks like, we will also look into the effect of the overlap region's size. We will do this simply by adding a next column to each of the subdomain. We will say. Below we can see a example of adding two columns to the subdomains after discretization, for $n = 4$.

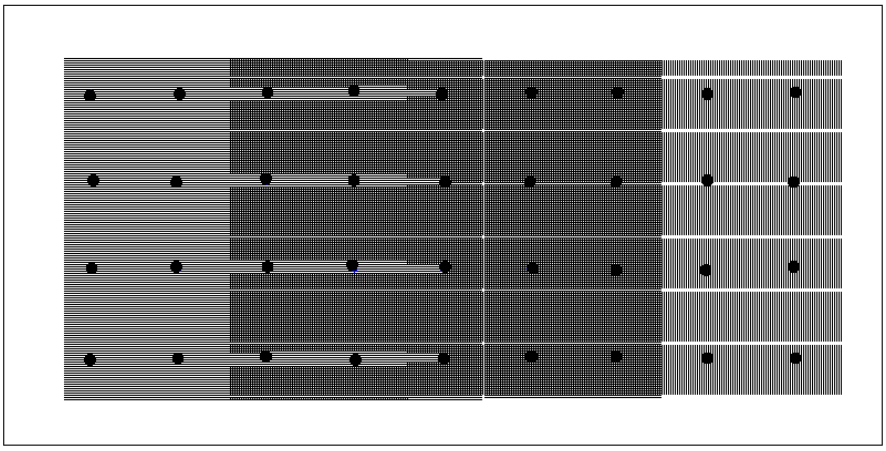

Figure 5.7: Domain Ω split into two subdomains, Ω_1 and Ω_2, with extended overlap of size 2.

We will say, that a subdomain has a extended overlap of size k, if it will consist of the base subdomain nodes and k additional columns.

5.4.1 Multiplicative Schwarz Method

First, we will look at the MSM method. We present how does the convergence behavior changes by increment of the overlapping region of two subdomains.

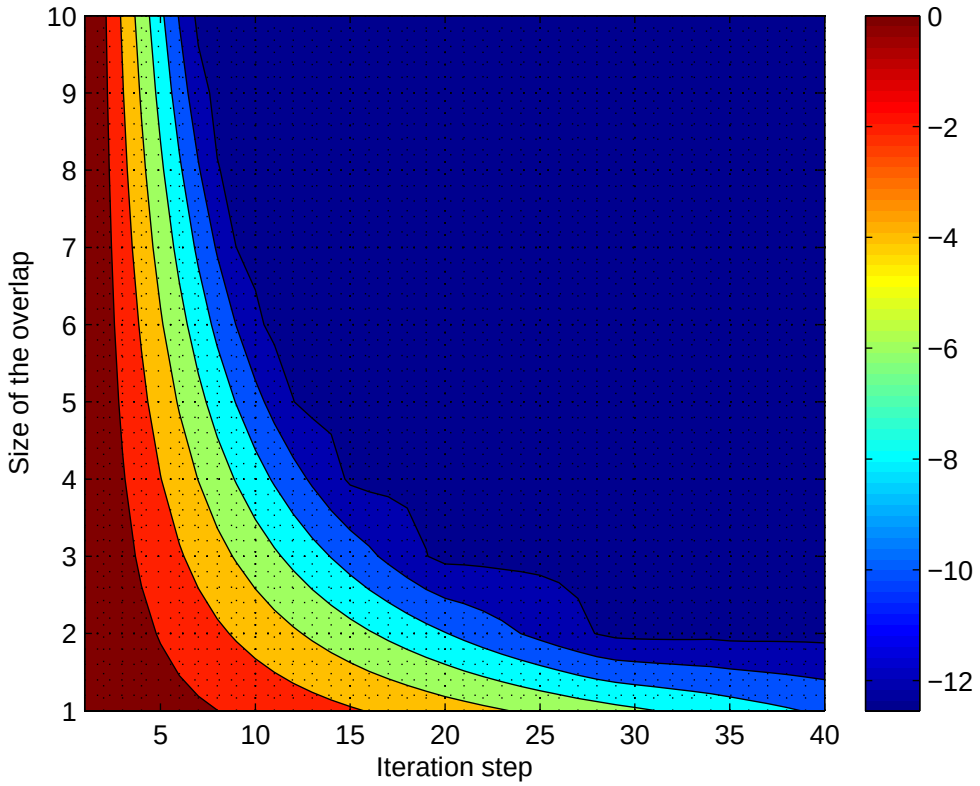

Figure 5.8: Contour plot of the log_{10} (i-th residuum) for MSM

The figure above shows us an essential information about the Multiplicative Schwarz Method. We see, that depending on the size of the overlap of two subdomains, we get faster or slover convergence of the method. We can notice, that the we are dealing here with a logarithmic dependency, between the size of overlap and number of iteration needed to achieve a certain error size.

5.4.2 Additive Schwarz Method

Now, we will present the result of the same experiment done for ASM.

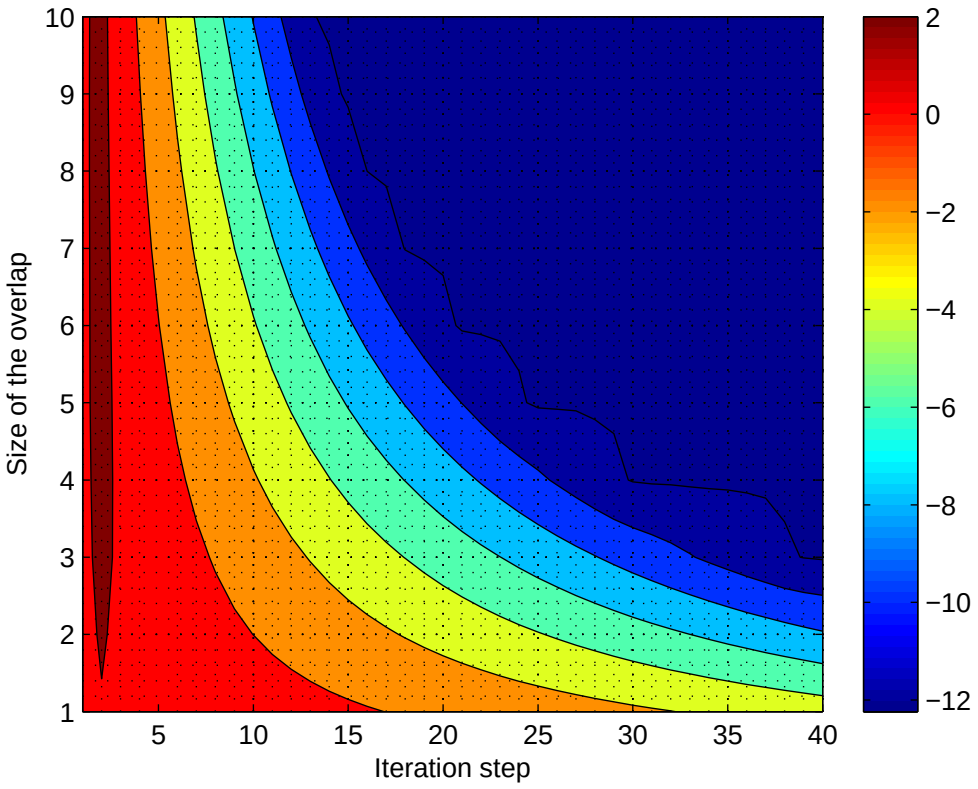

Figure 5.9: Contour plot of the log_{10} (i-th residuum) for ASM

As it was expected, we get a similar result as in MSM. Also here we have a logarithmic dependency between the number of iterations needed to achieve wanted error size and the size of overlap. However we may notice a slight slowdown of the convergence rate, which was expected, because ASM unlike MSM is computing the correction from each subdomain without any updates from neighboring subdomains. This is a drawback, however we can easily neglected it, because of the fact, that the corrections can be computed simultaneously, which can speeds up the whole processes in the sense of spent time.

We stop at this moment further investigation of the Schwarz method, due to the fact, that it is tested on a trivial problem. We will go deeper in the next chapter, when we will be working with test problems, provided by Plaxis.

Chapter 6

Research Goals

6.1 Introduction

Knowing all of the things which were presented so far in this report, we are at the moment, when we can actually start to work with finding a good parallel-friendly solver for linear systems which are generated by Plaxis software. However, to do that, we need to have some test problems, on which we will make numerical experiments. Also we need to say, what is actually our initial guess for the method, which we will to solve systems. And in the end, we should also define, what will be our next step in process of finding the solution to the problem.

6.2 Choice of the Method

In the very beginning of this project we have chosen, that a Domain Decomposition approach will be the starting point for solving the problem, due to the fact of the results noted in this field of mathematics. Nowadays, domain decomposition methods are getting more popular, because the underlying idea gives plenty of possibilities to parallelize the solution process, which is a wanted property when solving big systems.

From the big family of methods of this branch of mathematics, we have chosen to use one of the Schwarz Alternating Processes approaches, i.e the Additive Schwarz Method (ASM) as our solver. First reason for it was the fact, that the code can be easily converted into a parallel program. Secondly, there were already a try to use Schur Complement approach, however the results where not as good as it was expected. Also there are examples of many successful implementation of this method in real life problems, which only encouraged us to take ASM as our framebox for the solution of the linear systems.

Now when we have chosen ASM, to be our main core of the solver, we are going to upgrade it, by incorporating onto it two methods, which were presented in this report, i.e. Preconditioned Conjugate Gradient Method, for the solution of the linear systems of the subdomains, and the Deflation method to improve the spread of the information coming from a subdomain to other. This all combined together is going to be our tool in solving the linear systems.

The choice of the subdomains on which we will perform the ASM is going to be done not by a arbitrary cut of the coefficient matrix, but instead by adding the elements from the FEM process. In this way, we will have a possibility to incorporate

a intelligent partition of the domain, which will preserve the physical structure of the problem. We will see it in the following examples, which are going to be presented in this chapter.

6.3 Test Problems

In this section we will describe test problems, which were used as a starting point in finding the best solver for the linear system.

6.3.1 First Simple Problem

For the beginning of numerical experiments and tests, we have chosen to deal with a simple problem of volume displacement in a four layer cube, with a weight load situated on the top. Each layer has a different stiffness which implies jumps in the values of the coefficient matrix, corresponding to the nodes from different layers. The Finite Element discretization of the problem consist of 8 elements, 2 for each layer, and 61 nodes. This gives us in the end 76 degrees of freedom, due to the fact, that each node has it limitation of movement. We end up with a system of 77 degrees of freedom. Below we can see a illustration representing given problem.

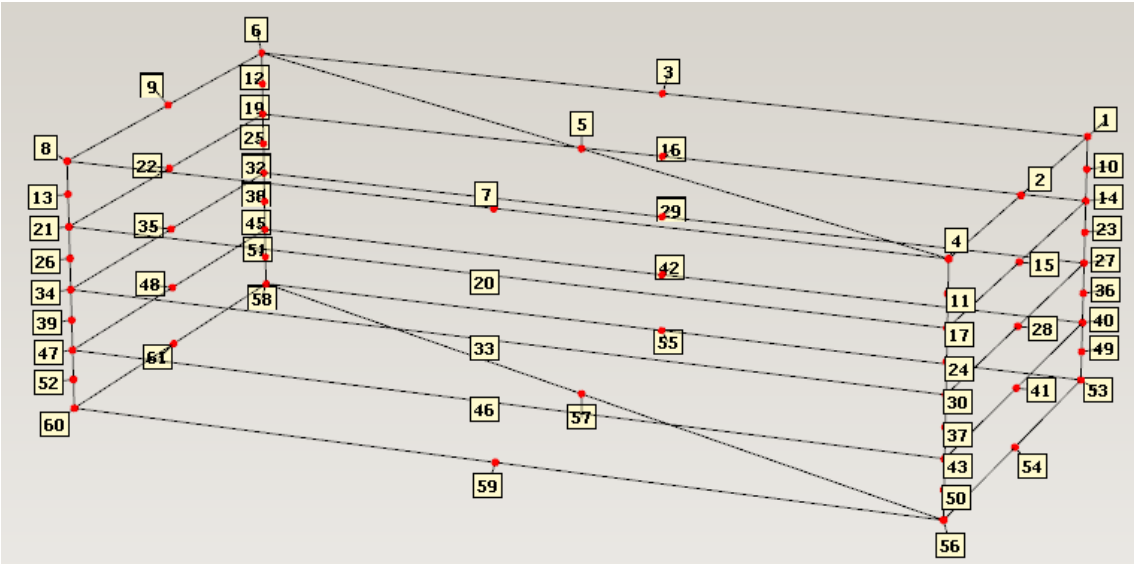

Figure 6.1: First Simple Problem Representation

The coefficient matrix of this problem has 2499 non-zeros elements.

We choose to cut the domain into 2 parts, namely the upper one, which will consist of two first layers from the top, and the second one, consisting of the three layers, taken from the bottom. The reason for that was, that we wanted to preserve similar sizes of two subdomains.

Let us first apply ASM to this problem with the subdomains defined as above, to see how does it react. As the result of this operation, we get a plot which shows us the rate of convergence of ASM method. We can clearly see from the plot below, that we are dealing here with a linear convergence behavior, which was expected because ASM is a Basic Iterative Method.

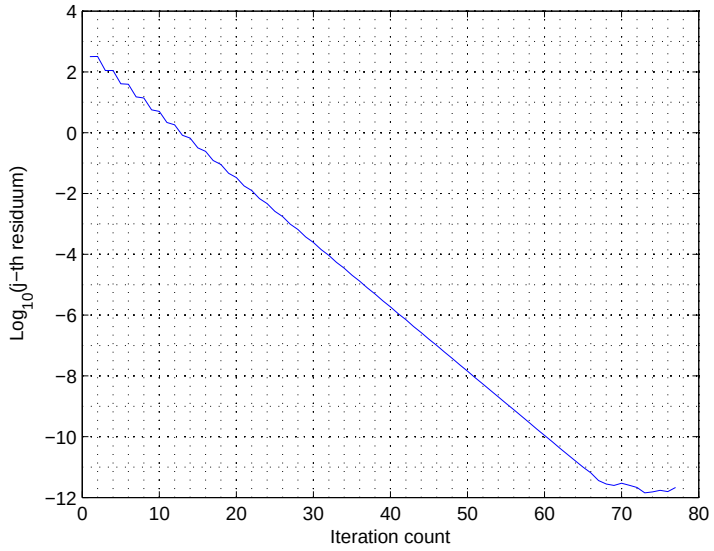

Figure 6.2: First Simple Problem convergence behavior for ASM

Now, we are going to put the PCG method into use for this problem. For the preconditioner we have taken ASM components, which were used in the previous experiment. As the result of this, we get a much faster convergence, then in the previous approach, what can be seen on the next page.

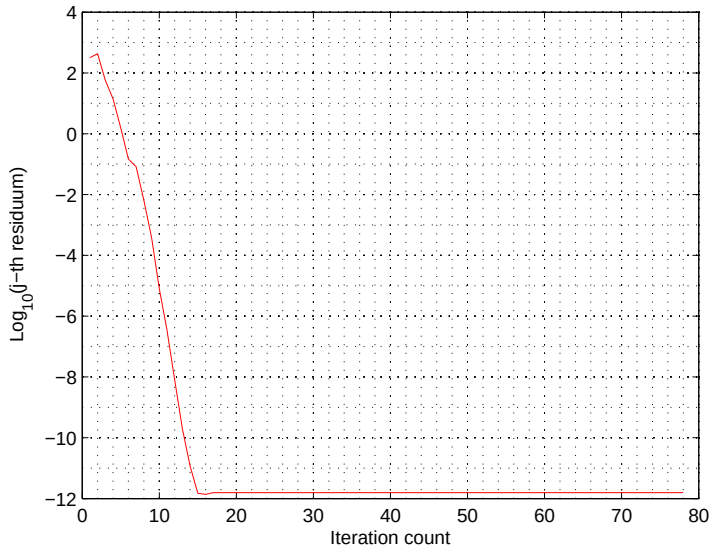

Figure 6.3: First Simple Problem convergence behavior for PCG

So far we have been looking into the behavior of convergence rate, when the subdomains are chosen via FEM elements. This decomposition of the domain implies, that the subdomains have an overlapping regions. It would be good also, to investigate the situation, when the subdomains are without it. For this purpose, we decided to create the subdomains algebraically, by simply spliting the coefficient matrix into two squared matrices along the diagonal, and neglect everything else. With this we get two can create a preconditioner and apply PCG to compute the solution of the system. Below we can find an illustration of the error distribution for each of the possible split for this problem.

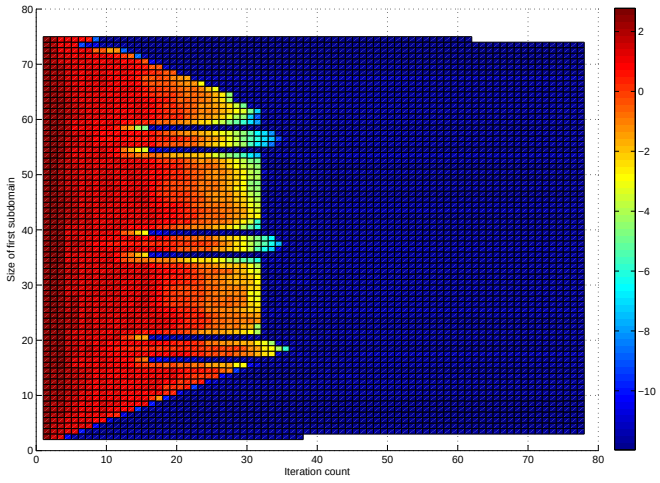

Figure 6.4: First Simple Problem distribution of error in PCG

The parabolic shape of the plot is quite logical. The bigger (smaller) is the factor of the split, the bigger one of the subdomains is getting to be, ergo is starting to be more similar to the whole domain, so the preconditioning part of it is more accurate. But more interesting are several dents, which occur in it. We can easily notice, that their positions are not random. They are correlated with the distribution of nodes in the elements from FEM.

6.3.2 Second Simple Problem

After dealing with the previous problem, there appeared natural need to extend it, to make it more complex. That is why we have chosen to make experiments on the same problem, but with a denser grid of nodes. The following picture present the new mesh which was used for generation of the system.

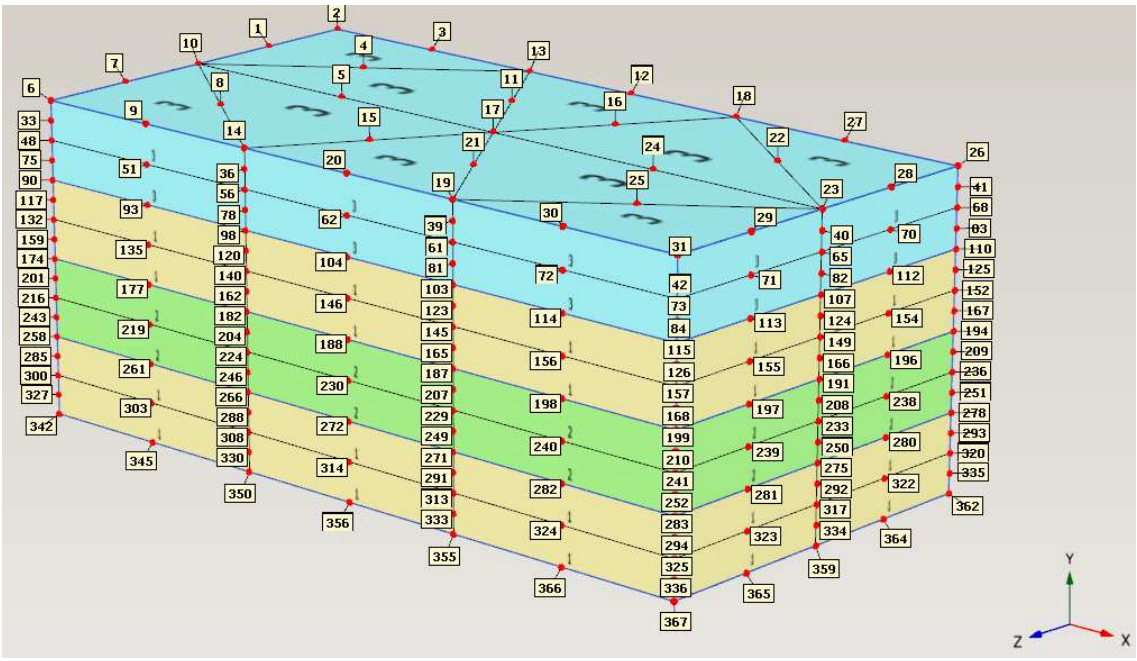

Figure 6.5: Second Simple Problem Representation

Now, the Finite Element discretization of the problem consist of 80 elements, 20 for each of the layer, and 367 nodes. We end up with system of over 705 degrees of freedom. We see, that the size of the problem is now almost 10 times bigger. The number of non zeros element in the coefficient matrix is 50483.

For this problem, we now define slightly different subdomains, i.e, we take for the first subdomain two first layers from the top, and for the second one, we take two instead of three, from the bottom. With such partition of the domain, we get two linear systems of a similar number of degrees of freedom.
On this structure, we apply now DPCG procedure. We decided to use Subdomain deflation vectors for it. On the next page we may see the convergence of this method for this problem.

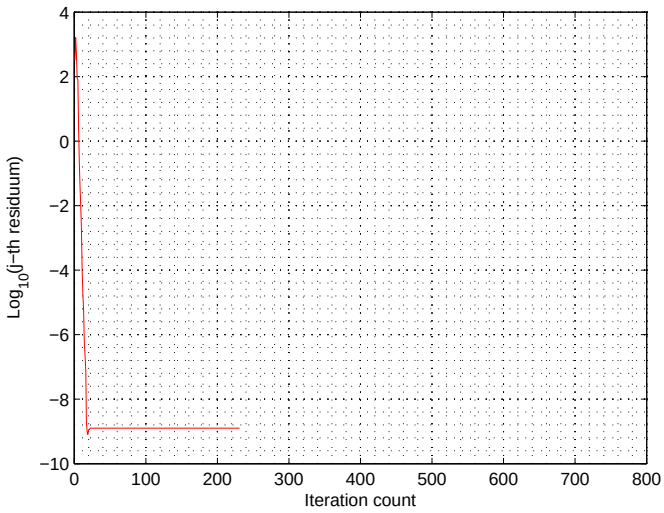

Figure 6.6: Second Simple Problem convergence behavior for PCG with 2 subdomains

We can clearly see, that for this problem the chosen method works perfectly. We get a error of size 10^{-4} in 11 iterations, what can be considered as a really small number if we take into consideration the size of the problem.
Till now, we have only considered decomposition into two subdomains. There arise a natural question, namely what will happen if we change the number of the blocks in which we split the domain. Let us now split the problem in four, instead of two subdomains, where each layer will be considered as a subdomain generator and apply our PCG method onto it.

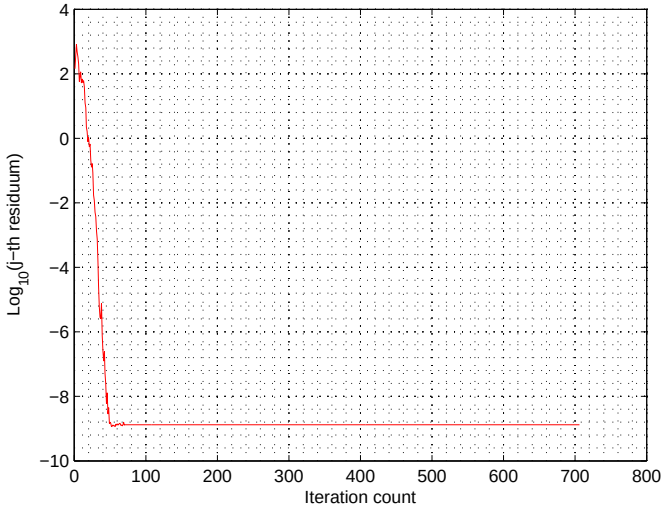

Figure 6.7: Second Simple Problem convergence behavior for PCG with 4 subdomains

The new decomposition makes the convergence rate slower in the sense of the iteration needed to get to the desired solution. However, we will investigate the time which was used spent for the computing the solution with the error size $10^(- 6)$, and we will consider that the computation of the block is done parallel by two processors, then the variant with four subdomains took around 0.13 second to get the result, while the variant with two spent 0.16 in average. We may see, that we were able to speed up the solution of the problem.
This result was a sparkle to check if we would split the problem into eight subdomains, we would also notice a speed up. For that, we split the domain, by simply taking each element layer and make it from it a subdomain used into precondtioning the system.

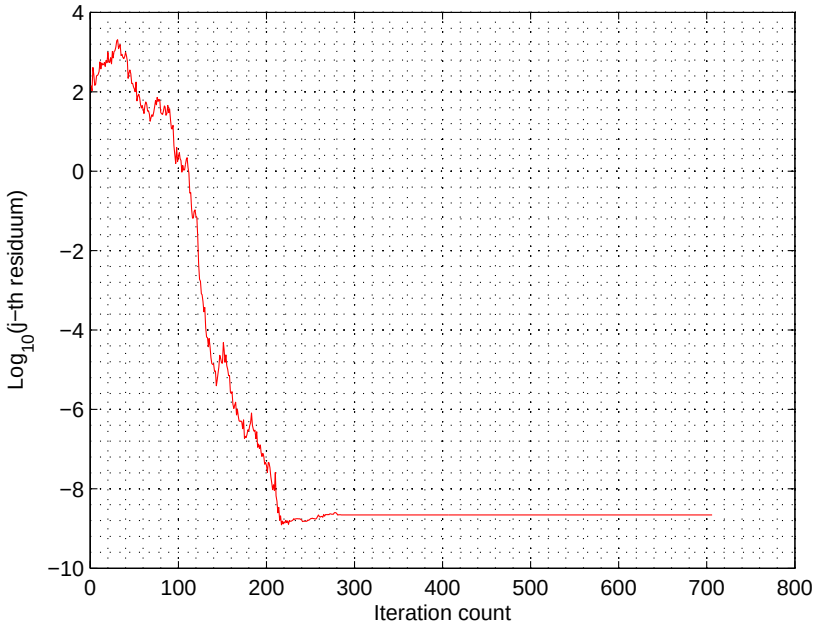

Figure 6.8: Second Simple Problem convergence behavior for PCG with 8 subdomains

The iteration number needed to get a descent error of the solution, is getting to grow with each increment of the number of subbdomains. We see that when having 8 subdomains, we need almost 200 iterations steps. This aspect transit into the amount of time spent for computing the solution. If we would considered, that the parts in the program where we were dealing with the subdomains would be done by two processors, then it would take almost 0.22 of second. We say almost, because comparing to the previous decomposition we notice a large jump.

We end up the experiments with the Second Simple Problem with a similar analysis done for the First Simple Problem, namely we would like to see how looks convergence rate, when we have non overlapping subdomains generated algebraically. Like before, we split the coefficient matrix into two squared matrices along the diagonal and used them to create a precondtioner used then in PCG.

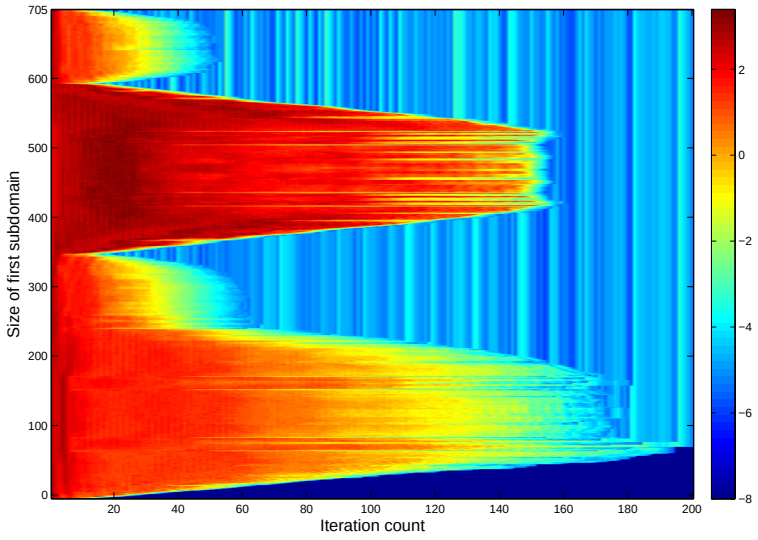

Figure 6.9: Second Simple Problem distribution of error in PCG

When simulating this, we decided that instead of inverting matrices used for the proconditioning, we will apply Cholesky Incomplete Factorization, to speed up the simulation. From the picture above we can clearly see, that the we can not cut the coefficient matrix by random. Once again we notice some places, which are more desired to use for the partition, then it's neighbors. However even more interesting is the fact, that when spliting at the 352 position, which tends to be the middle of the coefficient matrix, we have almost the best convergence behavior. It is interesting, that the position 354 is the beginning of the first element of the 3th layer from the problem. Also at postions 254 and 598 we notice a much faster convergence. Those nodes are actually the last nodes from the 1st and 3th later. So we clearly see, that the idea to decompose the problem with the incorporating underlying physics has a positive affect on the convergence behavior of the finding the solution.

6.4 Conclusions and research questions

From the test problems, which were presented in this chapter, we can clearly see that the approach which we have chosen to use in finding solutions for the linear systems is tend to be the the right one. We have seen that splitting cannot by done by simply cutting into two parts, because we may find that the structure of the problem can create some obstacles leading into a slower convergence. We have also seen, that incorporating elements from the discretization in the process of generation subdomains is a powerful tool in the acceleration of the speed of solution process.

However, we cannot forget that the ratio of number of subdomains and iteration steps plays also an important role in the process of finding the solution. In the second example, when we investigated this issue, we have noticed that we should also look at the amount of time used to solve the system. We should not only analyse the number of iteration steps, which can be misleading, because overall our goal is to decrease time spent on computation. This is our main research question at this moment, which is now strongly investigated.

We should also notice, that for now, we did not introduce any data which was solved by a method which uses Deflation approach. The reason for this is, that the results at the moment of writing this report where far of being satisfying. Nevertheless, there were done some numerical experiments, which have pointed the direction for solving this issue, which is namely the Rigid Body Modes approach as the choice of Deflation vectors.

Bibliography

[1] P. Bjorstad, B. Smith, W. Gropp, *Domain Decomposition.* Cambridge University Press: Cambridge, 1996.

[2] A. Cegielski (Editor), *Numerical Aspects in Applied Mathematics.* ZP UZm Zielona Gra, Poland, 2005.

[3] M. R. Hestenes,E. Stiefel, *Methods of Conjugate Gradients for Solving Linear Systems.* Journal of Research of the National Bureau of Standards Vol 49, No 6.

[4] J. van Kan, A. Segal, F. Vermolen, *Numerical Methods in Scientific Computing* VSSD, Delft, The Netherlands, 2005.

[5] Plaxis B.V., *Plaxis 3D Foundation: Scientific Manual version 2*, Delft, The Netherlands

[6] Y. Saad, *Iterative Methods for Sparse Linear Systems.* SIAM, Philadelphia, PA, USA, 2000. Second edition.

[7] A. Segal, C. Vuik, *Computional Fluid Dynamics II.* Delft, The Netherlands, 2006.

[8] K.H. Tan, *Local Coupling in Domain Decomposition.* Utrecht, The Netherlands, 1995.

[9] J.M. Tang, *Two-Level Preconditioned Conjugate Gradien Methods with Applications to Bubbly Flow Problems.* Delft, The Netherlands, 2008.

[10] A. Toselli, O. Widlund, *Domain Decomposition Methods - Algorithms and Theory* Springer-Verlag Berlin Heidelberg, 2005.

[11] J. Willie, *Internship report*, Vortech Computing, Delft, The Netherlands, 2008.

[12] E. Vollebregt, *De incomplete Cholesky preconditioner en de parallellisatie van Plaxis3D via OpenMP*, Memo EV/M08.029, version 1.1, VORtech, Juni 2008.

[13] T.B Jnsthvel, M.B. van Gijzen, C.Vuik, C. Kasbergen, A. Scarpas, *Preconditioned Conjugate Gradient Method Enhanced by Deflation of Rigid Body Modes Applied to Composite Materials*, CMES, vol.47, no.2, pp.97-118, 2009