Master's Thesis Project
Fibre-based preconditioner for granular matter simulation

Henrik Despaigne
c14hde@cs.umu.se

January 27, 2022

Abstract

Simulating granular media at large scales is hard to do because of ill-conditioning of the associated linear systems and the ineffectiveness of available iterative methods. One common way to improve iterative methods is to use a preconditioner which involves finding a good approximation of a linear system A. A good preconditioner will improve the condition number of A. If a linear system has a set of large eigenvalues of comparable magnitude, and the rest of the eigenvalues are small, so that the gap between the set of large eigenvalues and the small ones is large, the ill-conditioning caused by the small eigenvalues will not appear in the early iterations. We investigate a new fibre-based preconditioner that involves finding chains of contacting particles along the particles of a granular medium and reordering the system, which leads to a diagonal preconditioner. We show its effects on the relative residual and error of the velocity on linear systems where the ill-conditioning is caused by a big gap between a set of large eigenvalues and small eigenvalues for three different iterative methods: Uzawa, the Conjugate Residual (CR) and the Minimum Residual Method (MINRES).

Semester: VT 21
External Partner: Algoryx
Internal Advisor: Lars Karlsson
External Advisor: Claude Lacoursière

Contents

1 Introduction

Algoryx develops a physics engine called AgX Dynamics, which is their core product. For simulations, it is important to be able to model the real world as accurately as possible with proper physics. This requires good mathematical models which represent the physics accurately, but more than that, it also requires numerical methods which perform efficiently both for interactive, real-time applications and for offline simulations.

Granular matter form a vast family, from grains of corn to minerals, from pharmaceutical pills to pellets. Granular matter is composed of grains with different shapes and materials. At the grain level they are disordered, but at the macroscopic level, granular matter can behave like a solid or fluid and exhibits phenomena such as arching, avalanches and segregation [1].

As it stands today, large scale system experiments and measurements with granular media are almost impossible to conduct due to the ineffectiveness of the solution algorithms (solvers) that are available. Granular media is involved in many industrial and natural phenomena. In industry, problems of storage, transportation, flow and mixing often occur. In Earth sciences, as our soil is composed mainly of grains, phenomena such as sand dunes, earth slides, erosion patterns and pyroclastic flows can be observed [1]. Modeling and computer simulations are of great importance when it comes to understanding granular materials and for making improvements and innovations [2].

For simulations of granular matter, one widely used method is the discrete element method (DEM). In the late 1960's the discrete element method was developed by Cundall and Stracks [3]. It was developed as an extension of molecular dynamics using explicit time integration and smooth representation of contact forces to model macroscopic slightly deformable solid grains, and each particle is modeled as a rigid body. One of the major challenges of using DEM simulation is to reduce the computational time of large scale simulations [4].

When simulating granular matter with the discrete element method, the system of equations that is being solved is ill-conditioned, meaning that the condition number of the system is high which is not desired. It is undesirable since the solution to the system becomes hard to find and convergence is slow for iterative methods [5]. One of the approaches that has been identified to accelerate DEM simulations is to improve the convergence rate of the solver that is being used [2]. The ill-conditioning on the systems we are working with is caused by small eigenvalues that are many orders of magnitude smaller than the rest of the eigenvalues. We will explore the implications of ill-conditioning caused by these small eigenvalues.

By applying a preconditioner to an ill-conditioned system, it is possible to greatly improve the convergence rate for iterative methods solving the system. During the course of the project a *fibre-based preconditioner* has been developed and analysed. We will decide if such a preconditioner is viable by looking at the accuracy of the preconditioner for three different iterative methods: Uzawa, the Conjugate Residual and the Minimum Residual Method.

2 Background

In this section, already known and developed concepts will be described. The fibre-based preconditioner is work that is built on top of all the these concepts, and understanding them is important for understanding the work presented in this project.

2.1 Discrete Element Simulation

There are many different methods for simulating granular material. The one we are using is the discrete element method, which requires solutions of linear systems of equations. For a particle a in the DEM simulation, a has six degrees of freedom. It can be translated along the $x-, y-$ and $z-$axis, and rotated around the $x-, y-$ and $z-$axis, but for the scope of our thesis, we do not look at rotation because all our particles are spheres. Particle a has vectors containing a position $\vec{x}^{(a)}$, velocity $\dot{\vec{x}}^{(a)}$, force $\vec{f}^{(a)}$, torque $\vec{\tau}^{(a)}$, angular velocity $\vec{\omega}^{(a)}$, mass $m^{(a)}$ and inertia tensor $\mathcal{J}^{(a)}$. The velocity and angular velocity vectors are then concatenated into a generalized velocity vector, $v^{(a)}$ and the mass and inertia tensor vectors are made into a system state matrix $M^{(a)}$, where $v^{(a)} = (\dot{\vec{x}}^{(a)T}, \vec{\omega}^{(a)T})^T$ and $M^{(a)} = \text{diag}(m^{(a)} I_3, \mathcal{J}^{(a)} I_3)$. These are then represented as a full system state vector $v = (v^{(1)}, v^{(2)}, \ldots)$ and full system state matrix $M = \text{diag}(M^{(1)}, M^{(2)}, \ldots)$ [2].

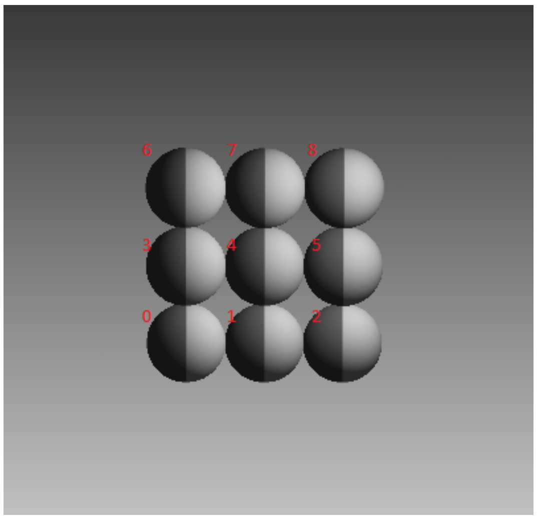
The *contact Jacobian* G is a matrix where each block row corresponds to one contact, and each block column is a particle. Each block row consists of either one or two non-zero blocks. If a row has one non-zero block it represents a contact between a particle and a geometry. If a row has two non-zero blocks it represents a contact between two particles. The non-zero blocks contains information about a specific contact for a particle. The two nonzero blocks $G_1^{(i)}$ and $G_2^{(i)}$ of $G^{(i)}$, that is, the two particles of contact i in G has the following structure

$$G_1^{(i)} = \begin{bmatrix} n^T & -(n \times p - p_1)^T \\ t_1^T & -(t_1 \times p - p_1)^T \\ t_2^T & -(t_2 \times p - p_1)^T \\ 0 & n^T \\ 0 & t_1^T \\ 0 & t_2^T \end{bmatrix}, \quad G_2^{(i)} = \begin{bmatrix} -n^T & (n \times p - p_2)^T \\ -t_1^T & (t_1 \times p - p_2)^T \\ -t_2^T & (t_2 \times p - p_2)^T \\ 0 & -n^T \\ 0 & -t_1^T \\ 0 & -t_2^T \end{bmatrix}, \tag{2.1.1}$$

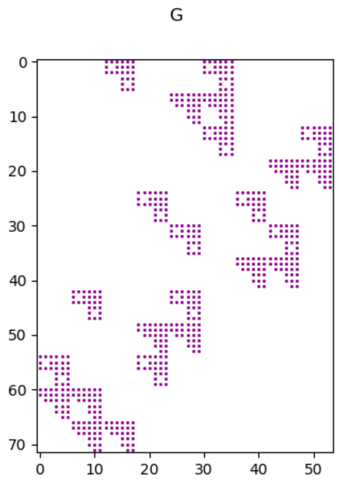
where n is the contact normal, t_1, t_2 are contact tangents, p is the contact point and p_1, p_2 are the positions of the two particles of the contact.

According to Newton's third law, we know that action is reaction, and from that it follows that $G_2^{(i)} = -G_1^{(i)}$. If the contact is a particle-geometry contact, then $G_2^{(i)} = 0$.

A three by three grid of particles would give the structure in the contact Jacobian which is illustrated in Figure 1



(a) Three by three particle grid



(b) Contact Jacobian

Figure 1 – Three by three particle grid and the resulting contact Jacobian.

If we look at any given column, we can see each contact that a particle is a part of. Block column one refers to particle zero, block column two to particle one and so on. If we look at any given row, we can see the particles that are a part of that contact. From the contact Jacobian we can see that the three by three grid has twelve contacts, which we can also verify by looking at the grid.

To compute the velocities for the next step of the simulation v_{k+1} of the particles, and to satisfy the constraints by computing the contact forces λ, we solve the linear system

$$\begin{bmatrix} M & -G^T \\ G & T \end{bmatrix} \begin{bmatrix} v_{k+1} \\ \lambda \end{bmatrix} = \begin{bmatrix} p \\ q \end{bmatrix}, \tag{2.1.2}$$

[4] where M and T are symmetric and positive-definite and $p = Mv_k + h$ and $q = \frac{1}{1+4\tau/h}(-\frac{4}{h}g_k + Gv_k)$ where h is the size of the time step. The dampening time for the stability of the time integration τ is defined as $\tau = 2h$, and g_k is the constraint violations of step k. $M \in \mathbb{R}^{n\times n}$ represents the mass of each particle in the system and is block diagonal and easily inverted. The contact Jacobian $G \in \mathbb{R}^{m\times n}$ is a sparse matrix, and usually $m \geq n$. We know that G is usually rank deficient in granular systems, as in most cases we have more rows than columns. The only time G is not rank deficient is when we have the same amount of particles as contacts. We have a perturbation $T \in \mathbb{R}^{m\times m}$ that is diagonal, where each entry on the diagonal is a compliance and the compliance is small. This perturbation corresponds to contact compliance and is used to ensure that the linear system in (2.1.2) is not degenerate, but it will be very ill-conditioned. For the velocities v_{k+1} we have the dimensions $v_{k+1} \in \mathbb{R}^n$, and for the constraint forces λ we have the dimensions $\lambda \in \mathbb{R}^m$. In solving (2.1.2) we are interested in the new velocities v_{k+1} and the constraint forces λ [6]. To compute the constraint forces, we have to compute the constraint violations. Constraint violations occur when bodies collide or are in contact

with each other. A constraint violation manifests as a *penetration depth* g between two bodies in a contact that describes how deep the bodies penetrate each other because of elasticity. To satisfy the constraints we have that $g(x) = 0$, that is, there is no penetration between two bodies in one time step. To achieve that, we need to compute the amount of force required to separate the bodies. This needs to be done for all contacts and for all particles. An illustration of a contact between two particles is shown in Figure 2.

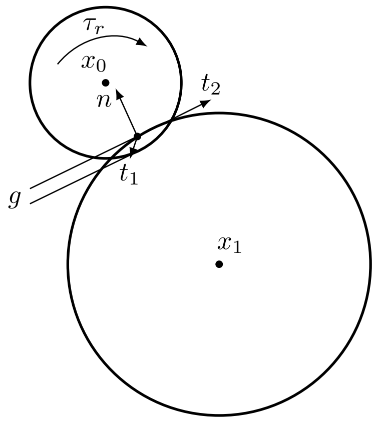


Figure 2 – Illustration of a contact between two particles x_0 and x_1.

2.2 Schur Complement

The Schur comeplement of a block matrix is a tool in linear algebra and theory of matrices that is defined as

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \tag{2.2.1}$$

where $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{m\times n}$, $D \in \mathbb{R}^{m\times m}$ and $M \in \mathbb{R}^{(n+m)\times(n+m)}$. If A is invertible. the Schur complement of the block A of matrix M is defined as

$$M/A = D - CA^{-1}B, \tag{2.2.2}$$

where $M \in \mathbb{R}^{m\times m}$. If D is invertible, then the Schur complement of block D of matrix M is defined as

$$M/D = A - BD^{-1}C, \tag{2.2.3}$$

where $M \in \mathbb{R}^{n\times n}$. A property of the Schur comeplement is that if we have that $B = C^T$ such that

$$M = \begin{bmatrix} A & C^T \\ C & D \end{bmatrix} \tag{2.2.4}$$

and if A is invertible, then M is positive-definite [7].

The matrix in (2.1.2) is not symmetric, which is an important property for some iterative methods. However, by taking the Schur complement of the matrix in (2.1.2) of block M, we get

$$A = GM^{-1}G^T + T. \tag{2.2.5}$$

which is symmetric and positive-definite. From this point forward, when we mention the Schur complement, we are specifically referring to the matrix in (2.2.5). An important connection to make is that $GM^{-1}G^T$ resembles structures found in other applications. It corresponds to the linear differential operator $\boldsymbol{\nabla} \cdot \left(\rho^{-1}\boldsymbol{\nabla}\cdot\right)$ of elliptic partial differential equations. This suggests that the alternate direction implicit method resembles our fibres [8]. However, if we do the Schur complement, we will not solve (2.1.2) anymore, but instead we will solve

$$A\lambda = q. \tag{2.2.6}$$

We are interested in solving for v_{k+1} as well, and it is possible to do that if we obtain λ. The equations of motion state that

$$Mv_{k+1} = Mv_k + hf + G^T\lambda, \tag{2.2.7}$$

which we can use to solve for v_{k+1} if we have solved for λ [9].

2.3 Krylov Subspaces

Some variation of a Krylov subspace method is commonly used in iterative methods that deal with large sparse matrices. Krylov subspaces are formed by applying a matrix to a vector time and time again. A Krylov subspace of order m is generated by a matrix $A \in \mathbb{R}^{n\times n}$ and a vector $v \in \mathbb{R}^n$ such that

$$\mathcal{K}_m(A, v) = \text{span}\{v, Av, A^2v, \ldots, A^{m-1}v\}. \tag{2.3.1}$$

The dimension of $\mathcal{K}$ increases by one every time we apply A to v. An important property of Krylov subspaces is that the vectors $\{v, Av, A^2v, \ldots, A^{m-1}v\}$ are linearly independent. Krylov subspace methods converge in maximum N iterations since these methods form a basis, where N is the size of the system. However the amount of iterations required to reach convergence can exceed N in the presence of rounding errors. For very large systems, the chosen iterative method can reach desired accuracy far earlier. Today problems are so large it is not feasible to run even N iterations [10].

2.4 Preconditioning

To find the solution to large sparse linear systems, it is in many cases beneficial to use iterative methods instead of direct methods. The benefits of iterative methods for these types of systems compared to direct methods are that they usually require less memory, they are easier to program and adapt to different types of problems and often a sufficiently accurate approximation to the solution is obtained with less computational effort [11]. Direct methods are also not practical for very ill-conditioned matrices or matrices that are nearly degenerate. For these types of matrices one would do Singular Value Decomposition (SVD), which is very slow [12].

Iterative methods, such as CG, often suffer from slow convergence in typical applications. The higher the *condition number* $\kappa(A)$ of the system matrix A is, the slower the convergence [10]. The condition number of a matrix A with respect to the norm is defined as

$$\kappa(A) = \|A\|\|A^{-1}\| \tag{2.4.1}$$

and it can give us a clue to how sensitive a linear system is. If A is symmetric and positive-definite, the condition number of A is defined as

$$\kappa(A) = \frac{\lambda_{\max}}{\lambda_{\min}}. \tag{2.4.2}$$

where $\lambda_{\max}$ is the largest eigenvalue, and $\lambda_{\min}$ is the smallest eigenvalue. The condition numbers of the matrices we work on throughout the project can reach as high as $\kappa(A) = 10^{12}$.

Another weakness with iterative solutions is lack of robustness. Since computers use finite precision arithmetic, rounding errors can lead to numerical errors and loss of important properties in the algorithms. Both of these weaknesses can be improved by using preconditioning [10].

Preconditioning is simply the process of transforming a linear system into another linear system with the same solution but with an improved condition number. We follow the exposition in Shewchuk and let P be a matrix that is assumed to be more easily invertible than a matrix A and is symmetric and positive-definite. We can then solve

$$P^{-1}Ax = P^{-1}b \tag{2.4.3}$$

and indirectly get the solution for $Ax = b$. This is called a *left preconditioned* system. However, $P^{-1}A$ is not symmetric and positive-definite, but it is similar to a positive-definite system. Since M is positive-definite it can be factored as $P = EE^T$, with Cholesky for example. We can find a similarity transformation from $P^{-1}A$ to $E^{-1}AE^{-T}$. Here we have that $E^{-1}AE^{-T}$ is symmetric and positive-definite and has the same eigenvalues as $P^{-1}A$. The standard preconditioned version is equivalent to working with that system, therefore solving for (2.4.3) is equivalent to solving

$$(E^{-1}AE^{-T})(E^T x) = E^{-1}b. \tag{2.4.4}$$

However, we do not want to compute E, and to circumvent this, it is possible to eliminate E with variable substitutions and only work on P. If $\kappa(E^{-1}AE^{-T}) \ll \kappa(A)$ or the eigenvalues of $E^{-1}AE^{-T}$ are better clustered than those of A, we can solve (2.4.3) iteratively faster than the original problem [11].

There are two common types of preconditioners: incomplete factorizations and approximate inverses. The main difference between the two are that incomplete factorization computes P and P^{-1} is applied but not formed explicitly. Where the other P^{-1} is directly built and applied. The fibre-based preconditioner is an incomplete factorization [13].

In the context of iterative methods, a good preconditioner P should be cheap to compute and it should be able to solve (2.4.3) faster than the original problem. A preconditioner that satisfies these criteria will lead to faster and more accurate solutions.

The best approximation of the original system would be $P = A$, as $\kappa(P^{-1}A) = 1$. However this would mean that we would have to solve the system $Px = b$ for the preconditioning step, which renders this preconditioner useless [11].

2.5 Conjugate Gradient Solver

The conjugate gradient method, first proposed by Hestenes and Stiefel [14], is a Krylov subspace method used to solve linear systems of the form

$$Ax = b \tag{2.5.1}$$

where x is unknown, and A and b are known. For CG, A needs to symmetric positive-definite. For A to be positive-definite we have that

$$\hat{x}^T A \hat{x} > 0, \tag{2.5.2}$$

for every nonzero vector $\hat{x}$.

For iterative methods we generally have a *residual* $r_{(i)} = b - Ax_{(i)}$ that tells us how far we are from the correct value of b at iteration i. For each iteration, an iterative method produces a *candidate solution* x^*, while the *stationary point* x is the actual value of the solution.

The CG algorithm uses the Method of Conjugate Directions. The idea is to pick a set of A-orthogonal search directions d, where two vectors $d_{(i)}$ and $d_{(j)}$ are A-orthogonal, or *conjugate*, if

$$d_{(i)}^T A d_{(j)} = 0. \tag{2.5.3}$$

The CG algorithm will take n steps, and take exactly one step in each search direction with step length $\alpha_{(i)}$ [11]. Let us start with the basic CG algorithm with the matrix A in (2.2.5). We are solving for x in $Ax = b$. First we compute the step length for iteration i, and then take a step in the current search direction with the computed step length which updates our candidate solution. We then update our residual and then do the conjugation step and compute β, which will ensure us that our next search direction will be $A-$orthogonal to our other search directions. Finally we compute the next search direction. We repeat this process until our residual is below a desired tolerance ϵ. Doing this will result in Algorithm 1.

Algorithm 1: Conjugate Gradient

```
   Data: $r_{(0)}, \epsilon$
   Result: $x_{(\nu+1)}$
1  $d_{(0)} \leftarrow r_{(0)}$ ;
2  $\nu \leftarrow 0$ ;
3  while $r_{(\nu)} > \epsilon$ do
4      $\alpha_{(\nu)} \leftarrow \frac{\|r_{(\nu)}\|^2}{d_{(\nu)}^T A d_{(\nu)}}$ ;
5      $x_{(\nu+1)} \leftarrow x_{(\nu)} + \alpha d_{(\nu)}$ ;
6      $r_{(\nu+1)} \leftarrow r_{(\nu+1)} - \alpha A d_{(\nu)}$ ;
7      $\beta_{(\nu+1)} \leftarrow \frac{\|r_{(\nu+1)}\|^2}{\|r_{(\nu)}\|^2}$ ;
8      $d_{(\nu+1)} \leftarrow r_{(\nu+1)} + \beta_{(\nu+1)} d_{(\nu)}$ ;
9      $\nu \leftarrow \nu + 1$ ;
10 return $x_{(\nu+1)}$
```

2.5.1 Preconditioned Conjugate Gradient

CG, as well as most other Krylov subspace methods, are not used without preconditioning. These types of algorithms perform poorly without preconditioning and does not converge well. With a good preconditioner, the convergence of CG and other Krylov subspace methods can drastically improve. Instead of solving for $Ax = b$, we can indirectly solve for (2.4.3). The *Untransformed Preconditioned Conjugate Gradient Method* showed in Algorithm 2 solves for the preconditioned system [11].

Algorithm 2: Untransformed Preconditioned Conjugate Gradient Method

```
   Data: $r_{(0)}, \epsilon$
   Result: $x_{(\nu+1)}$
1  $d_{(0)} \leftarrow P^{-1} r_{(0)}$ ;
2  $\nu \leftarrow 0$ ;
3  while $r_{(\nu)} > \epsilon$ do
4      $\alpha_{(\nu)} \leftarrow \frac{r_{(\nu)}^T P^{-1} r_{(\nu)}}{d_{(\nu)}^T A d_{(\nu)}}$ ;
5      $x_{(\nu+1)} \leftarrow x_{(\nu)} + \alpha d_{(\nu)}$ ;
6      $r_{(\nu+1)} \leftarrow r_{(\nu+1)} - \alpha A d_{(\nu)}$ ;
7      $\beta_{(\nu+1)} \leftarrow \frac{r_{(\nu+1)}^T P^{-1} r_{(\nu+1)}}{r_{(\nu)}^T P^{-1} r_{(\nu)}}$ ;
8      $d_{(\nu+1)} \leftarrow r_{(\nu+1)} + \beta_{(\nu+1)} d_{(\nu)}$ ;
9      $\nu \leftarrow \nu + 1$ ;
10 return $x^{(\nu+1)}$
```

We never compute the inverse P^{-1} of the preconditioner. Instead we solve a linear system $Ps = r \implies s = P^{-1}r$, which can then be used to initialize $d_{(0)}$ as well as compute α and β. An important note to make is that even if preconditioning can improve the convergence of the algorithm, it does not necessarily mean the algorithm performs better in real time measurement. The preconditioned algorithm requires one to solve a linear system for each iteration, which costs time. A preconditioner does not necessarily improve convergence either, a bad preconditioner can even impair convergence [10].

2.5.2 Uzawa Algorithm

We do not want to explicitly compute A in (2.2.5). The reason for this is because we know that G is sparse and usually very large. As it often is when one performs matrix multiplication with sparse matrices, they become dense, which is not desirable for computational efficiency. A also has to be stored somewhere in memory, and it is so big we can not realistically store it.

Instead of computing this explicitly, we can split the computation of Ax into two parts. The equation to solve then becomes

$$x_{(\nu+1)} = Ax_{(\nu)}. \tag{2.5.4}$$

We know the Schur complement A from (2.2.5). We can rewrite Ax as $GM^{-1}G^Tx + Tx$. To compute this we can write it as

$$w = M^{-1}G^Tx \tag{2.5.5}$$

and

$$u = Gw + Tx \tag{2.5.6}$$

where w and x are dummy variables and will be replaced by the true unknowns λ and v later.

We can modify Algorithm 1 to arrive at the *Uzawa algorithm* [15] as shown in Algorithm 3.

Algorithm 3: Uzawa Algorithm with conjugate directions

```
Data: $r_{(0)}, \epsilon$
Result: $x_{(\nu+1)}$
1  $d_{(0)} \leftarrow r_{(0)}$ ;
2  $\nu \leftarrow 0$ ;
3  while $r_{(\nu)} > \epsilon$ do
4      $z \leftarrow M^{-1}G^T d_{(\nu)}$ ;
5      $\gamma \leftarrow z^T M z$ ;
6      $\rho \leftarrow d_{(\nu)}^T T d_{(\nu)}$ ;
7      $\alpha \leftarrow \frac{w}{\gamma+\rho}$ ;
8      $\lambda \leftarrow \lambda + \alpha d_{(\nu)}$ ;
9      $v \leftarrow v + \alpha M^{-1} f$ ;
10     $r \leftarrow r - Gv + T\lambda - q$ ;
11     $w_1 \leftarrow \|r\|^2$ ;
12     $\beta \leftarrow \frac{w_1}{w}$ ;
13     $d \leftarrow r + \beta d_{(\nu)}$ ;
14 return $x_{(\nu+1)}$
```

The important modifications done in this algorithm compared to Algorithm 1 is that we do not explicitly compute A. For the scope of this thesis, we compute $G^T d$ and Gv on line 2 and 8 with matrix-vector multiplications by explicitly forming G. However, if this was to be used in practice, we would compute this implicitly without forming G. Instead of doing matrix-vector multiplications we would implement kernels which can be used to do these computations without doing the matrix-vector multiplications, as well as computing γ, ρ and w in the kernels. The benefit of doing it with the kernels rather than the explicit computation is better performance and less memory usage. If memory usage can be reduced to a point where all the required memory for the computations can fit in the CPU's L3 cache, then considerable performance improvement can be achieved.

2.6 Conjugate Residual

The Conjugate Residual method (CR) is a Krylov subspace method and it is closely related to CG. While CG only works on symmetric, positive-definite matrices, the only requirement for CR is that the system matrix is Hermitian (equal to its own conjugate transpose). The downside of CR compared to CG is that it requires one additional vector update ($2n$ more operations), and one more vector of storage. CG and CR typically display similar convergence [10].

One issue that CG has, but CR does not, is that the residual in CG is not monotonically decreasing, which causes problems with early termination. This is because for any iterate, the candidate solution may be closer to the real solution, but the true residual might be larger. This leads to the constraint error changing unpredictably between iterations. CR does not suffer from this issue, since the residual in CR is monotonically decreasing.

2.7 Minimum Residual

The Minimum Residual Method (MINRES) is a Krylov subspace method. Just like CR, MINRES operates on symmetric linear systems, while CG requires the system to be symmetric and positive-definite, MINRES works on indefinite systems as well. MINRES can be preferable over CG if it is desired to terminate early on positive-definite systems. The residual for MINRES is often smallar than for CG by an order or two of magnitude. The downside of MINRES is that it is more computationally expensive [16].

3 Fibre Preconditioning

Using the information available about a matrix and exploiting its structure are valuable tools when trying to find a good preconditioner for said matrix. The idea of the fibre preconditioner is to change the structure of the contact Jacobian in such a way that a lot of the important information in the Schur complement end up close to the diagonal, and then extract a preconditioner as a block diagonal matrix. The idea is a continuation of previous work by Mattias Linde on cloth simulation. Linde produced promising results on grid-like structures as shown in Figure 3.

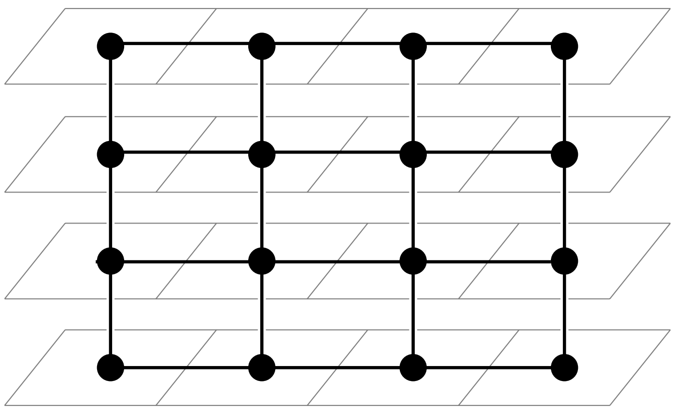


Figure 3 – Four by four particle grid.

He managed to get these promising results by first finding chains horizontally and then vertically in the grid as shown in Figure 4. When finding horizontal chains he would relabel the particles in an incrementing manner where the first particle got labeled as particle zero, second particle as particle one and so on. When finding the vertical chains, no particle labeling will be done since all the particles are already relabeled in the horizontal sweep. The contacts are also relabeled on both the horizontal and vertical sweeps.

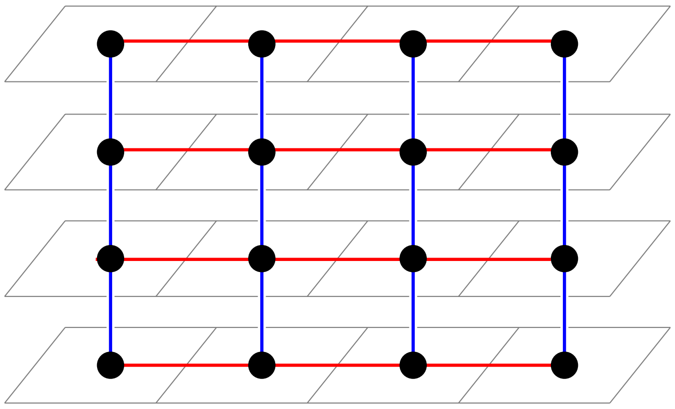


Figure 4 – An example on how Linde picked the chains. He first took the horizontal chains (red) and then the vertical chains (blue). Doing this changes the structure of the contact Jacobian and the Schur complement.

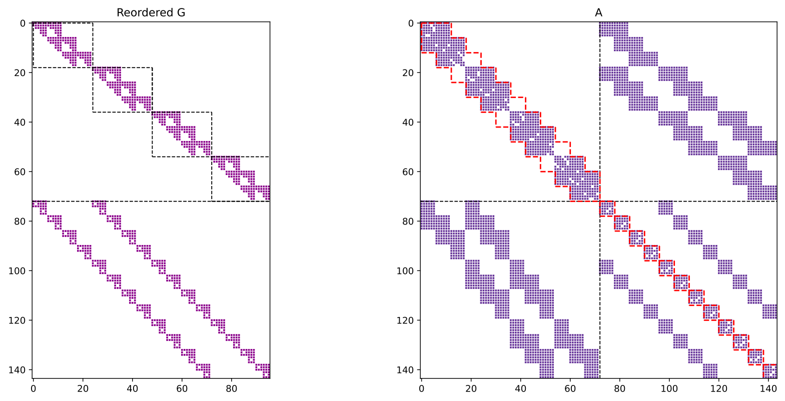


Figure 5 – The left plot shows the structure of the contact Jacobian after reordering the way Linde did. The right side shows the resulting Schur complement. As we can see in the contact Jacobian we have a block structure where each block has a block bi-diagonal.

The reason the particles are relabeled in such a way is to get the blocks for each contact in the chain to be adjacent to each other in the contact Jacobian. The reason the contacts are relabeled as well is to get the bi-diagonal block structure as we can see in the upper part of the contact Jacobian in Figure 5. When the Schur complement A is constructed the blocks of bi-diagonals gets centered in the top left block of A as seen in A in Figure 5.

To obtain the preconditioner, he then extracted the block tri-diagonal from A. We can see what part of A this would correspond to in Figure 5 where the block tri-diagonal is marked in red. He saw that reordering the contact Jacobian this way, it produced a preconditioner that made the residual decay rapidly [17]. The obvious downside to this approach is that it only works on grid-like structures. We would want something that works on any type of topology.

The fibre preconditioner involves finding chains of adjacent particles of a granular medium and relabeling the particles and contacts in an incrementing manner. These chains are what we call the fibres. The fibre preconditioner will try to replicate the bi-diagonal blocks as much as possible in the contact Jacobian in Figure 5. A fibre consists of a set of two or more particles, where all adjacent particles in the fibre are in contact with each other. Unlike Linde, we do not restrict ourselves to only taking particles in a certain direction in each fibre. We do put a a condition that a particle a can only belong to a single fiber. How many particles there are in one fibre, or how many fibres will be constructed is random and will be determined by the topology of the system and the starting point of a fibre. A fibre will try to consist of as many particles as possible. When a fibre can not take any more particles, the fibre ends and the construction of a new fibre starts at a new particle that is not in any fibre. To then extract the preconditioner, we extract the block diagonal with blocks of size six from the resulting Schur complement. We do not extract the block tri-diagonal like Linde did, because for our tests it did not improve the preconditioner any noticeable way compared to just the block diagonal. In Figure 6 we can see an example of how one single big fibre is created and the contacts that do not make it into the fibre end up under the block diagonal. In Figure 7 we can see the resulting contact Jacobian and Schur complement.

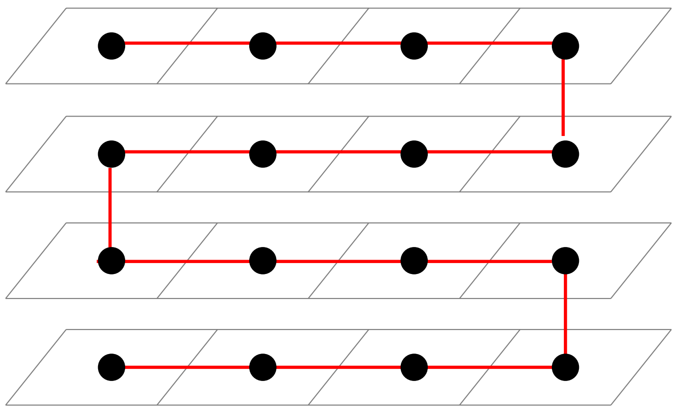


Figure 6 – Example of how one big fibre can be created in the four by four grid case.

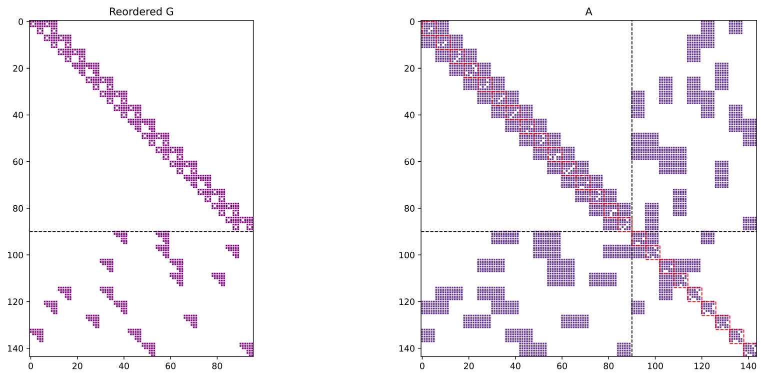


Figure 7 – The left plot shows the structure of the contact Jacobian after reordering with the fibres. The right side shows the resulting Schur complement. We can see that one big fibre was created because we have one long bi-diagonal block.

In Figure 8 we can see an example of how two fibres are created. In the contact Jacobian in Figure 9 we can also see how the structure changes when we get new fibres. When a new fibre starts we can see that we start building a new bi-diagonal block.

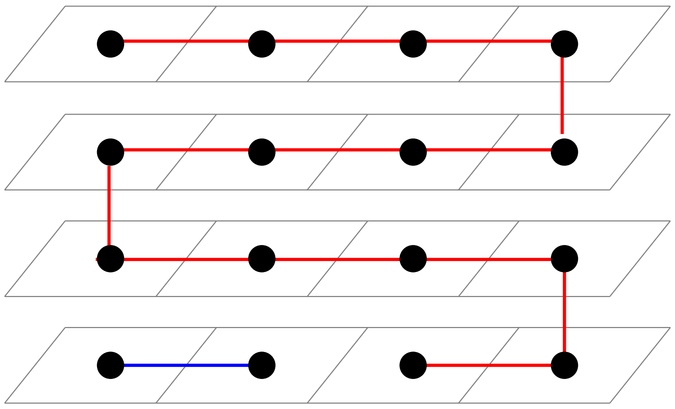


Figure 8 – Example of how two fibres can be created in the four by four grid case. If we start at the third particle from the left at the bottom row and work our way up to the top left particle, we will not be able to take any more particles into the fibre. We then start over at a new particle we have not taken and go from there.

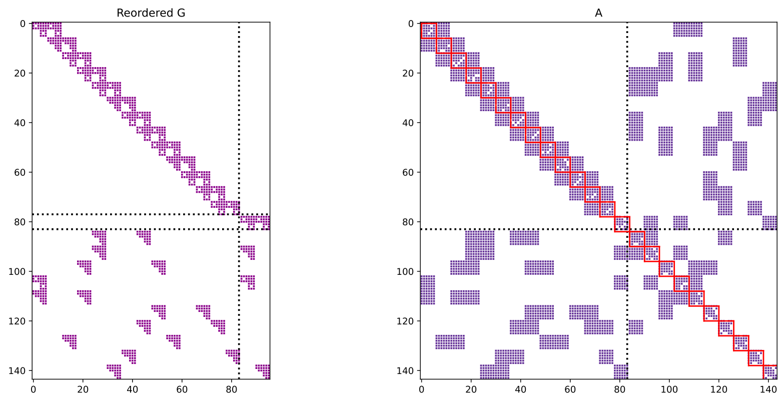


Figure 9 – The left plot shows the structure of the contact Jacobian after reordering with the fibres. The right side shows the resulting Schur complement. We can see that one big fibre and one small fibre was created.

We also have cases where not all particles are taken. On big systems this case is the most common one. In Figure 10 we can see a case where all particles will not be taken into a fibre. The way this manifests in the contact Jacobian is that the blocks of bi-diagonals does not span across all the columns, as we can see in Figure 11. This will leave us with a zero block on the top right block of the contact Jacobian.

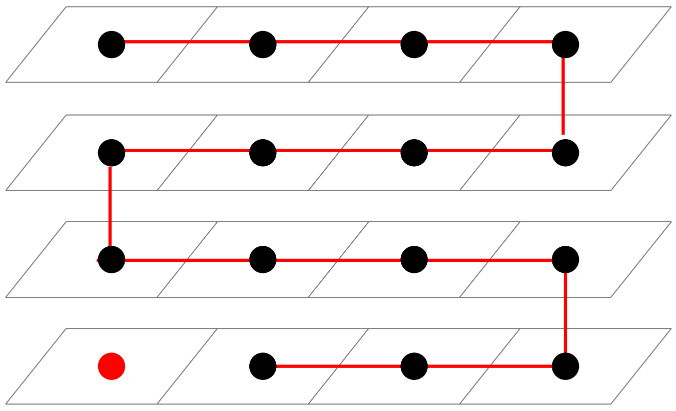


Figure 10 – Example of how we will not take all the particles in the four by four grid case. If we start at the second particle from the left at the bottom row and work our way up to the top left particle, we will not be able to take any more particles into the fibre. However when we try to start on a new fibre we discover that all adjacent particles are taken, and we terminate without all particles belonging to a fibre.

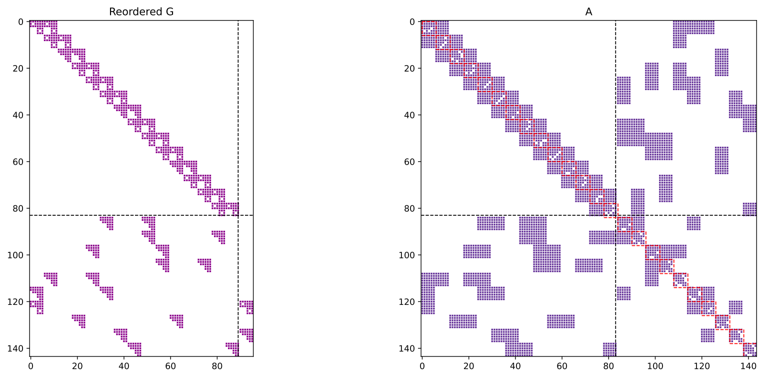


Figure 11 – The left plot shows the structure of the contact Jacobian after reordering with the fibres. The right side shows the resulting Schur complement. We can see that not all particles were taken into the fibre, as if it did we would have a bi-block diagonal spanning through all the columns of the upper block. Instead we get a zero block at the top right block.

An important thing to note is that even if the structure has changed from the reordering, it is still the exact same system of equations. Generating the fibres is a graph theory problem, where the particles are nodes and the contacts are edges. To generate a fibre, we start at an arbitrary node. To determine which edge we will take to continue the fibre we put weights on the edges. We do this so we can try to replicate how Linde selected his horizontal chains, and by putting weights we can try to force this behaviour. The weight of an edge is the absolute value of the difference in the z position of the two particles of a contact. To determine what edge to take, we take the edge with the smallest weight. Every node and edge we take gets marked, and are never taken again. By taking the smallest weight, the algorithm will search for new nodes to add to the fibre where the nodes are on a similar height in the z position. The algorithm shown in Algorithm 4 shows how to generate one fibre by sending in a starting particle, as well as a set of marked particles and contacts. The sets keep track of what particles and contacts we have taken before, so if we start on a new fibre, we still know which particles and contacts we can not take. In this algorithm we also save the the contacts and bodies in a way to permute them in the contact Jacobian later, however as we will discuss later, this is not the optimal thing to do. For better performance, the contacts and particles should be relabled as we are constructing the fibres, so ideally this algorithm should not return anything, just create the fibres.

Algorithm 4: Fibre Algorithm

```
Data: current_particle, marked_particles, marked_contacts
Result: contact_permutations, particle_permutations
if current_particle in marked_particles then
    return
while current_particle not None do
    particle_permutations.add(current_particle);
    SET $l$ equal to EMPTY LIST;
    for contacts of current_particle do
        $z \leftarrow$ abs ($z$_particle - $z$_particle in contact) ;
        if particle in contact is not in marked_particles and contact is not in
          marked_ contacts then
            $l$.add($z$, contact_particle, contact);
    if $l$ is empty then
        current_particle = None;
    else
        current_particle $\leftarrow$ contact_particle of lowest $z$ in $l$;
        contact_permutations.add(contact of lowest $z$);
return contact_permutations, particle_permutations
```

This will in 2D-like structures, like grids, roughly correspond to the way Linde reordered his structure, since we will only be taking horizontal edges as far as we can, until there are no more horizontal edges to take, then a vertical contact will be taken and the procedure will repeat. In 3D topologies, the algorithm will sweep across each plane, taking as many particles in the fibre as it can before moving to the next plane. This is illustrated for the topology in Figure 12, one possible way the algorithm could take the fibres for this problem is shown in Figure 13.

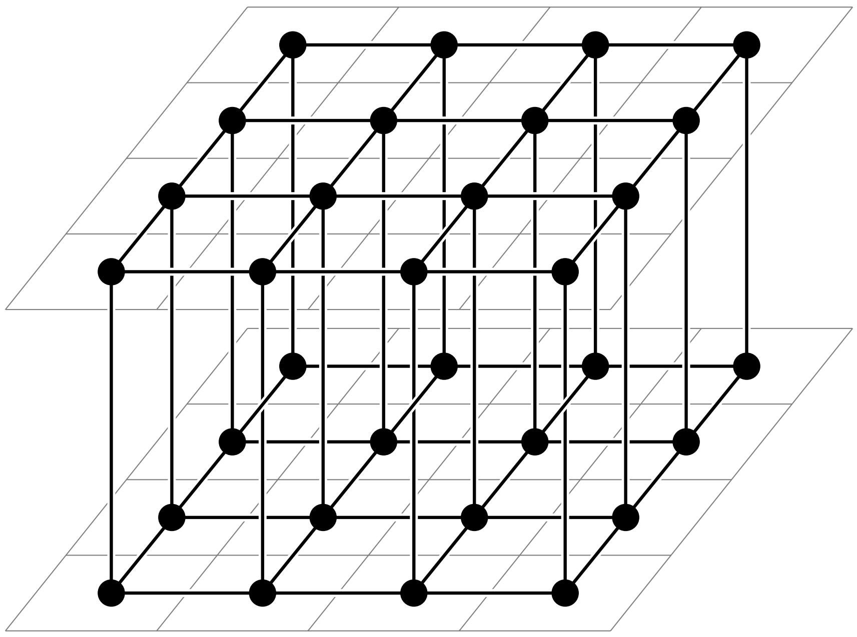


Figure 12 – An example with two layers of particles.

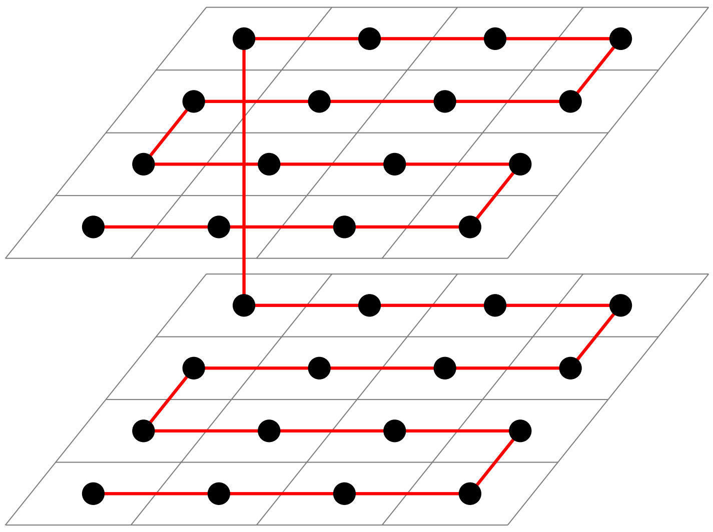


Figure 13 – One way the fibres can be selected. The algorithm takes as many particles it can on the current plane before moving to the next plane.

In Figure 14, we can see a contact Jacobian and the reordering procedure on a bigger and unstructured example. This example better highlights that we will most likely not take all particles, as can be seen from the blocks of bi-diagonals not spanning across all columns. This means that the top right zero block of the reordered matrix grows larger as more particles do not end up in a fibre.

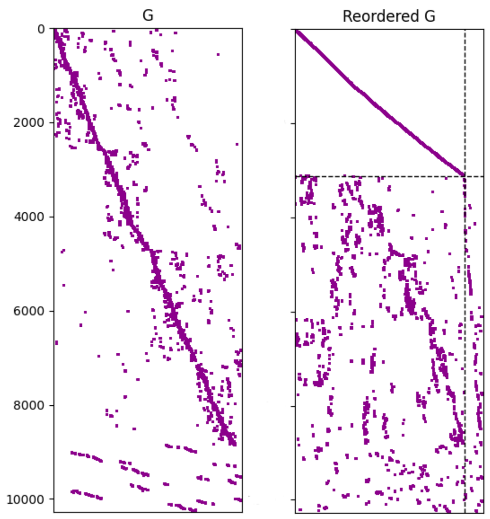


Figure 14 – Example of how a contact Jacobian and the fibre reordering of the same contact Jacobian can look at a bigger and unstructured example.

4 Methodology

To ensure that the implementations of our iterative methods are correct we have used Scipy as a reference [18]. Scipy is a library that is widely used for scientific and technical computing and is based on well tested and trusted methods from Netlib [19]. For Uzawa we have used Scipy's implementation of CG as a reference, and for CR we have used Scipy's implementation of MINRES as reference. MINRES and CR performs equally on positive-definite matrices.

5 Testing Methodology

For the numerical experiments, we will conduct two types of tests. The first test is called the *silo test* which we use because the silo test is a standard test for granular matter simulation. For the silo test, a particle emitter that creates particles of different sizes and masses is placed above a cone-like funnel, and when the particles exit the funnel they get removed. When the test starts, the particles will pour out of the funnel at a slower rate than they pour in, which leads to the funnel clogging up. The second test is called the *material test*. For this test, a particle emitter that creates particles of different sizes and masses is placed above a plane. When the test starts, the particles will fall down on the plane and eventually pile up.

We will perform each type of test five times each with different amount of particles. We will analyse the convergence rates of the relative residual. We use the relative residual as measurement since the decay of the relative residual is dependent on the size of the data, and this will give us a good idea of how the convergence is affected by the size of the problem. The relative residual is defined as

$$r_r = \frac{\|r\|}{\|A\|}. \tag{5.1}$$

The eigenvalues of the linear system we want to solve will also be analyzed for each test. Looking at the eigenvalues is key to understand the behaviour of the convergence. We will use Numpy's [20] function *eigs* to compute the eigenvalues of the linear system A and preconditioned system $P^{-1}A$. The eigs function is implemented using LAPACK routines [21], LAPACK routines are also well tested and trusted.

5.1 Error of the velocity

Another measurement we will look at is the error of the velocity. Recall from (2.2.7) that we can compute the velocity of the particles for the next step of the simulation v_{k+1} if we have computed the contact forces λ. The reason we use the error of the velocity as a measurement of the solution instead of something like the forward error, where the forward error is defined as $\|x - x^*\|$ is because the contact Jacobian G is rank deficient. If we do a *QR factorization* of G we get

$$G = QR = Q \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix}. \tag{5.1.1}$$

Here we have that $Q_1 \in \mathbb{R}^{m\times n}, Q_2 \in \mathbb{R}^{m\times(m-n)}$ and $R_1 \in \mathbb{R}^{n\times n}$. We get this QR factorization because G is a rectangular matrix with more equations than variables. Now let us construct the Schur complement

$$\begin{aligned} A = GM^{-1}G^T + T &= \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix} M^{-1} \begin{bmatrix} R_1^T & 0 \end{bmatrix} \begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix} + T \\ &= \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 M_{11}^{-1} R_1^T & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix} + T \end{aligned} \tag{5.1.2}$$

Since we know that T is a diagonal matrix with all elements on the diagonal being $t = 1e-8$, and Q is orthogonal by definition from the QR factorization, we can write

$$T = tI = tQQ^T = QTQ^T. \tag{5.1.3}$$

Rewriting T in this way, we can then arrive at

$$\begin{aligned} A &= \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 M_{11}^{-1} R_1^T & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix} + \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} T \begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix}, \quad \text{from (5.1.2), (5.1.3)} \\ &= \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 M_{11}^{-1} R_1^T + T_{11} & 0 \\ 0 & T_{22} \end{bmatrix} \begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix}. \end{aligned} \tag{5.1.4}$$

We define

$$H = \begin{bmatrix} R_1 M_{11}^{-1} R_1^T + T_{11} & 0 \\ 0 & T_{22} \end{bmatrix}, \tag{5.1.5}$$

which then implies that

$$A = QHQ^T. \tag{5.1.6}$$

Now we can write our system of equations as

$$\begin{aligned} Ax &= (QHQ^T)x = QHy, \quad Q^T x = y \\ &= \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 M_{11}^{-1} R_1^T + T_{11} & 0 \\ 0 & T_{22} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = b. \end{aligned} \tag{5.1.7}$$

By multiplying (5.1.7) by Q^T from the left we arrive at

$$\begin{aligned} (R_1 M_{11}^{-1} R_1^T + T_{11}) y_1 &= (Q^T b)_1 \\ T_{22} y_2 &= (Q^T b)_2. \end{aligned} \tag{5.1.8}$$

We can now define the residual as

$$\begin{aligned} r &= b - Ax \\ &= b - QHy, \quad \text{from (5.1.7)} \\ &= Q(Q^T b - Hy). \end{aligned} \tag{5.1.9}$$

The norm of the residual is then

$$\|r\| = \left\| \begin{bmatrix} (Q^T b)_1 - (R_1 M_{11}^{-1} R_1^T + T_{11}) y_1 \\ (Q^T b)_2 - T_{22} y_2 \end{bmatrix} \right\|. \tag{5.1.10}$$

From (5.1.7) we let y be a matrix-vector multiplication by some matrix Q^T and x. We can then write the forward error as

$$\|x - x^*\| = \|y - y^*\|, \tag{5.1.11}$$

where $y^* = Q^T x^*$ and $y = Q^T x$. We now have a term $\|y - y^*\|$ that depends on the forward error.

Finally, we know that the residual can be expressed as a term of the forward error:

$$\begin{aligned} \|r\|^2 &= \|A(x - x^*)\|^2 \\ &= \|H(y - y^*)\|^2 \\ &= \left\| \begin{bmatrix} H_{11}(y_1 - y_1^*) \\ T_{22}(y_2 - y_2^*) \end{bmatrix} \right\|^2 = \|H_{11}(y_1 - y_1^*)\|^2 + \|T_{22}(y_2 - y_2^*)\|^2. \end{aligned} \tag{5.1.12}$$

Here we can see that even if the residual is small, we can have a high forward error. If $\|r\|$ is small, then $\|y_2 - y_2^*\|$ can be large because we know that $T_{22} = 1e - 8$ is small.

Thus it is possible to have a large forward error even with a small residual. If we do a QR factorization of G^T in (2.2.7), as we did on G in (5.1.1), we get that

$$\begin{aligned} G^T\lambda \iff & \left(\begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix}\right)^T \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} \\ & = \begin{bmatrix} R_1^T & 0 \end{bmatrix} \begin{bmatrix} Q_1^T\lambda_1 \\ Q_2^T\lambda_2 \end{bmatrix} = \begin{bmatrix} R_1^T Q_1^T \lambda_1 & 0 \end{bmatrix} \sim \begin{bmatrix} G_1^T & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix}. \end{aligned} \tag{5.1.13}$$

This suggests that parts of λ, specifically λ_2 will not affect the computed value of v_{k+1}. This is the reason why we choose the measure the error of the velocity instead of the error of the solution we get from the iterative methods. From (2.2.7) we define the error of the velocity as

$$v_e = \|(v + M^{-1}f + M^{-1}G^T x^*) - (v + M^{-1}f + M^{-1}G^T x)\|. \tag{5.1.14}$$

5.2 Effective Condition Number

We will also look at the *effective condition number* of the original and the preconditioned system. We will define exactly what this means, but first we need to look at some properties of our systems and the implications of these properties.

If A has n eigenvalues, of which r of them are large and the remaining $n-r$ eigenvalues are small, and we let the eigenvalues of A be ordered such that $\lambda_1 \geq \lambda_2 \geq \ldots \lambda_n > 0$, we can define the small eigenvalues as $\lambda_{r+i} \leq \epsilon\lambda_r$, where $0 \leq \epsilon \ll 1$, and $i = 1, 2, \ldots, n-r$. If we have a set of large eigenvalues of comparable magnitude, and the rest of the eigenvalues are small, it has been shown for CG that the residual associated with the large eigenvalues are made small first [22], without changing the residual associated with the small eigenvalues that much. Forsgren proves this by showing that the iterates of the large eigenvalues are similar to those of another system that only has the same set of large eigenvalues. In addition to this, he also shows that the iterates associated with the small eigenvalues are close to the initial residual. Forsgren shows that the ill-conditioning of the problem caused by the small eigenvalues does not appear until the residual associated with large eigenvalues are made small. Forsgren's analysis can also be applied to CR by minimizing the function $x^T A^2 x - 2b^T Ax$ instead.

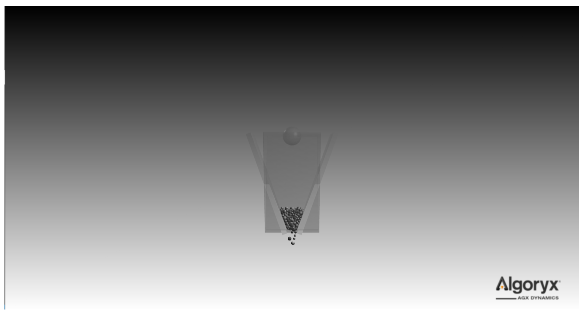
We can now define what we mean with the effective condition number. Instead of looking at the condition number, we look at the effective condition number, where define the effective condition number as the condition number of the large eigenvalues only. That is, for a linear system A, the effective condition number $\kappa_e(A)$ of A is defined as

$$\kappa_e(A) = \frac{\lambda_1}{\lambda_r}. \tag{5.2.1}$$

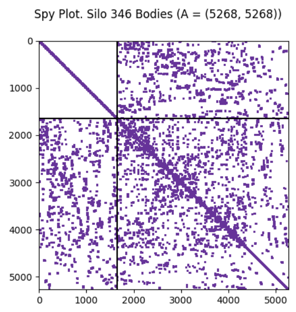
However as stated earlier, this analysis only applies if the large eigenvalues are of comparable magnitudes. As we will see later, this will definitely be true for the preconditioned systems, but not necessarily for the original systems.

6 Numerical Experiments

The first experiment we will look at is the silo test for 346 bodies, as seen in Figure 15.



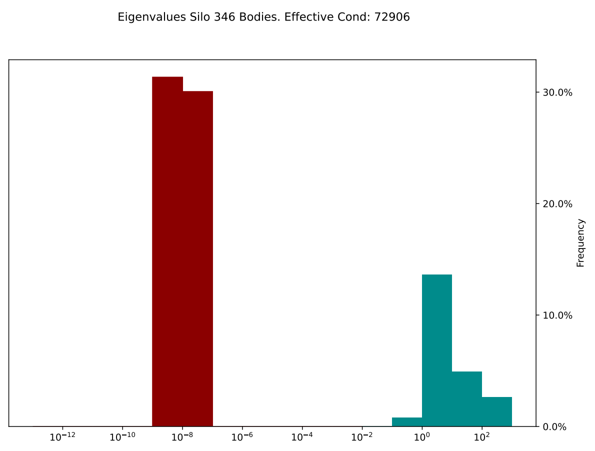
(a) Silo test with 346 bodies.



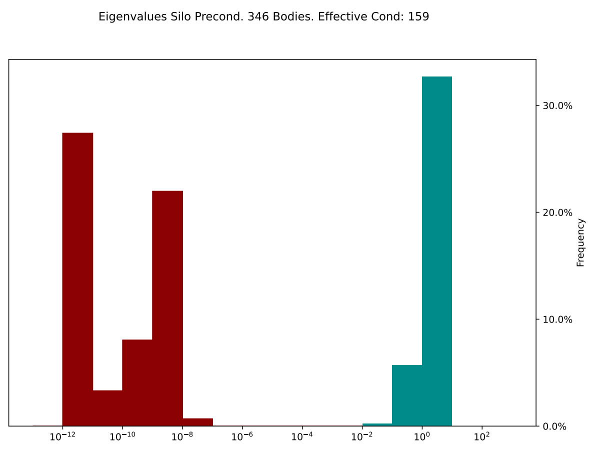
(b) Schur complement of the time step we are looking at.

Figure 15 – The scene at the timestep we are running our tests on and the Schur complement.

We first look at the eigenvalues of the original and preconditioned system as shown in the histograms in Figure 16. The red bins represents the small eigenvalues and the other bins represents the set of large eigenvalues. From the histograms we can see that for both the original and the preconditioned system, we have a set of large eigenvalues, while the rest are small and there is a large gap between the small and the large eigenvalues. For the preconditioned system, the large eigenvalues are all of comparable magnitude, so we know that we can look at the effective condition number here. The same is not necessarily true for the original system, where the orders of magnitude of the large eigenvalues differ. We still measure the effective condition number of this system and put it as a best case scenario of the condition number of the original system. The effective condition number of the original system is $\kappa_e(A) = 72906$ and $\kappa_e(P^{-1}A) = 159$ for the preconditioned system. If we did measure the regular condition number this would be in the order of magnitude of 10^{12}. One thing to note for the small eigenvalues of the original system is that we know these eigenvalues already, without using any algorithm to compute them. The small eigenvalues in the original system comes from the perturbation we add to ensure our system is not degenerate, and these eigenvalues are all $\lambda = 10^{-8}$. As of now, we do not know any possible way to make these small eigenvalues better.



(a) Histogram of the eigenvalue clustering for the silo test with 346 bodies.

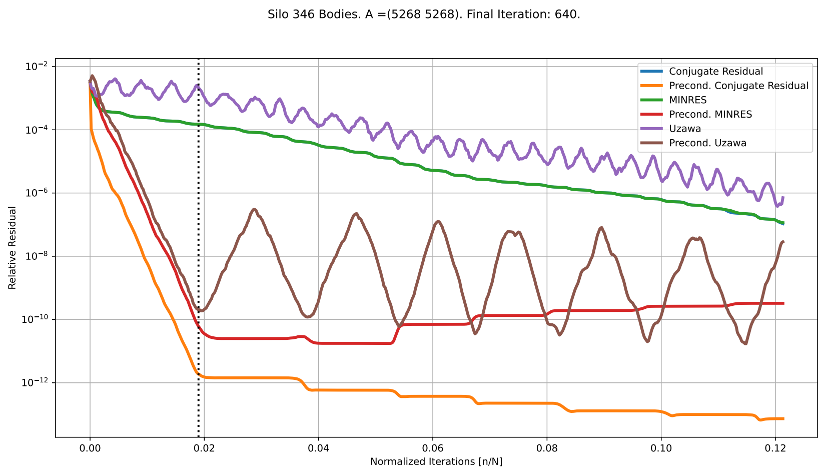


(b) Histogram of the eigenvalue clustering for the preconditioner for the silo test with 346 bodies.

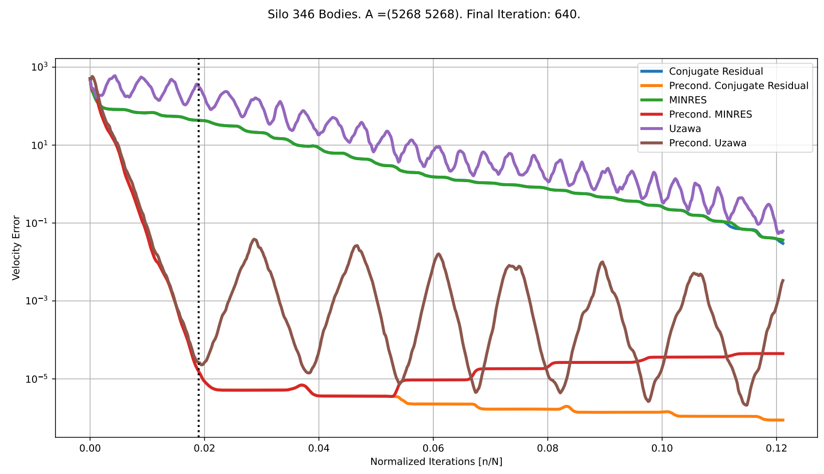
Figure 16 – Histograms over the eigenvalue clusters for the silo test with 346 bodies.

From the convergence curve in Figure 17 we can see that the preconditioner performs well for the early iterations. Once the residual associated with all the large eigenvalues have been made small and we move on to the small eigenvalues, we can see a dramatic shift in the convergence, and we stop converging instantly and stagnate. By the final early iteration the relative residual for the preconditioned system is up to eight orders of magnitude smaller. By the final iteration the change is minimal from the final early iteration due to the stagnation. We plot a vertical dotted line at 100 iterations for visual aid, this helps visualizing roughly how many absolute iterations we have performed and not just how many iterations we have performed in relation to the size of the system.

The error of the velocity also looks good, as we see a similar decay in the error for the early iterations, and then a complete stagnation once we switch over to the small eigenvalues. The performance of the three methods are roughly the same, as is expected. We can see that CR and MINRES perform exactly the same for the large eigenvalues, and Uzawa performs equally for most of the large eigenvalues.



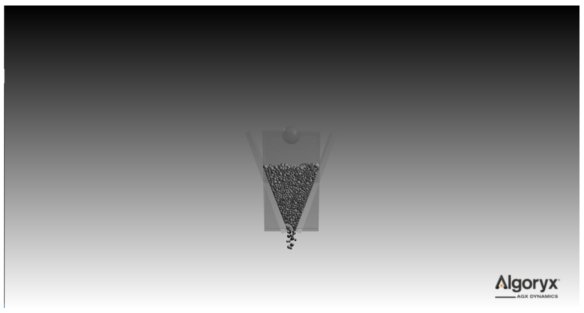
(a) Convergence plot for the relative residual for the silo test with 346 bodies.



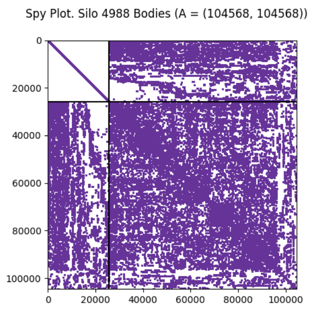
(b) Convergence plot for the error of the velocity for the silo test with 346 bodies.

Figure 17 – Convergence plots for the silo test with 346 bodies.

The second experiment we will look at is another silo test, but this time with 4988 bodies as seen in Figure 18.



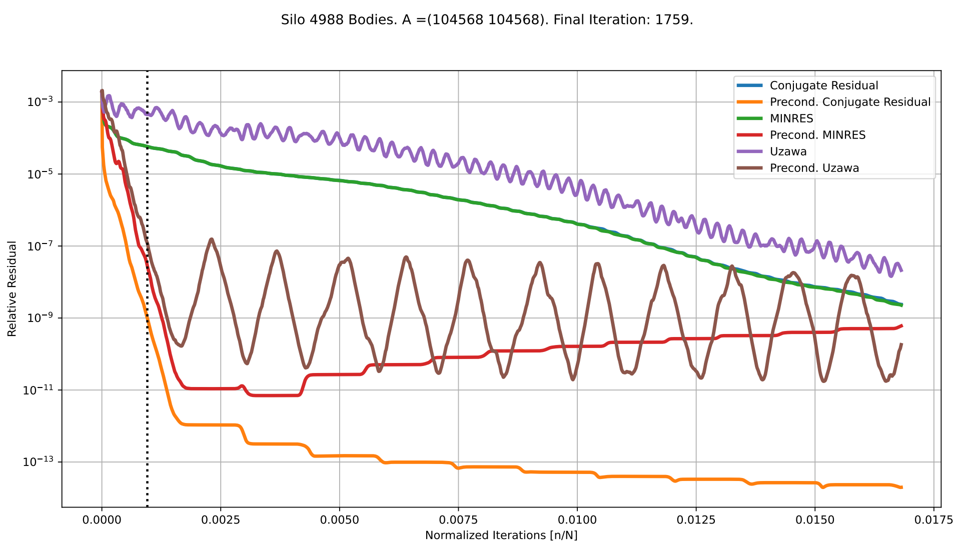
(a) Silo test with 4988 bodies.



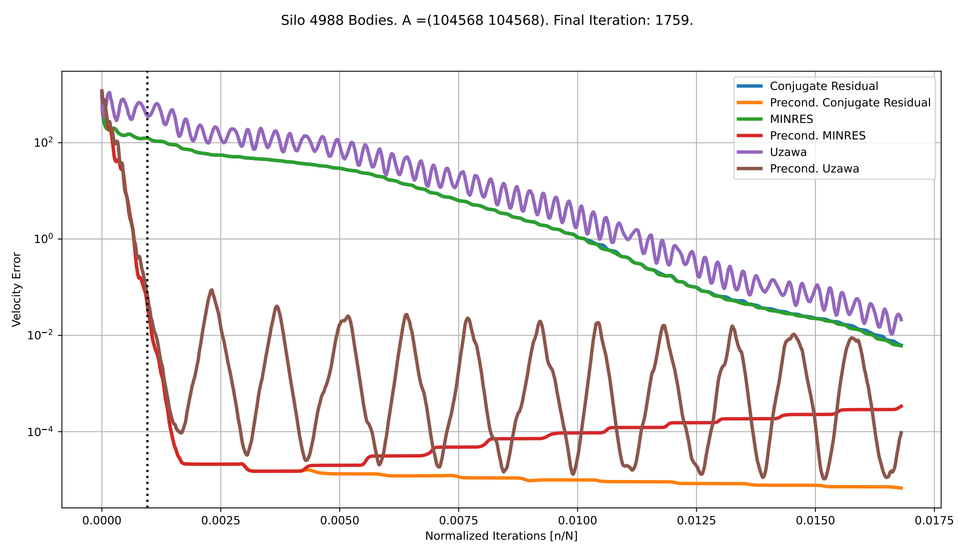
(b) Schur complement of the time step we are looking at.

Figure 18 – The scene at the timestep we are running our tests on and the Schur complement.

We will not look at the eigenvalue histograms for this test, for the simple reason that on the workstation the tests were performed, it was not possible to compute all the eigenvalues for the size of this test. It is possible to compute all but one eigenvalues for sparse matrices with the Scipy library, but since that leaves out one eigenvalue which can be anything, we do not compute the eigenvalues this way. If we look at the convergence plots in Figure 19, we see similar behaviour to the convergence of the silo test with 346 bodies with a steep decrease during the early iterations and then it stagnates.



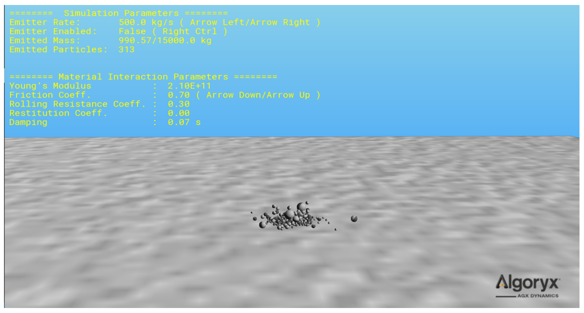
(a) Convergence plot for the relative residual for the silo test with 4988 bodies.



(b) Convergence plot for the error of the velocity for the silo test with 4988 bodies.

Figure 19 – Convergence plots for the silo test with 4988 bodies.

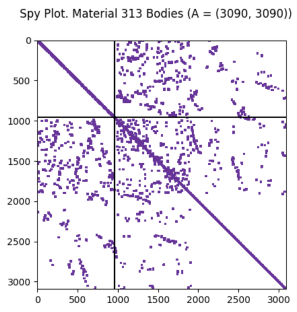
The third experiment we will look at is the material test for 313 bodies as seen in Figure 20.



```
========  Simulation Parameters ========
Emitter Rate:        500.0 kg/s ( Arrow Left/Arrow Right )
Emitter Enabled:     False ( Right Ctrl )
Emitted Mass:        990.57/15000.0 kg
Emitted Particles:   313


======== Material Interaction Parameters ========
Young's Modulus              :  2.10E+11
Friction Coeff.              :  0.70 ( Arrow Down/Arrow Up )
Rolling Resistance Coeff.    :  0.30
Restitution Coeff.           :  0.00
Damping                      :  0.07 s
```

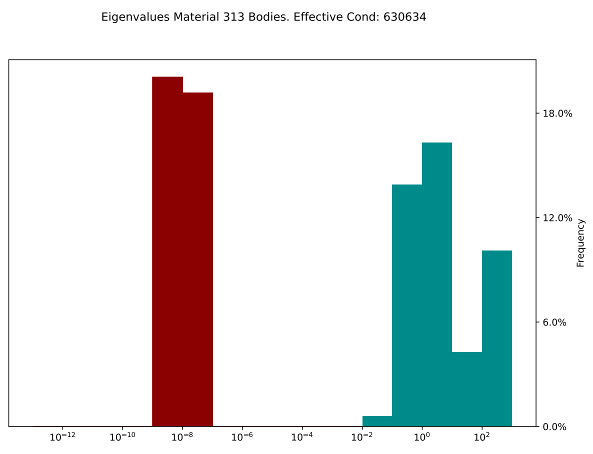
(a) Material test with 313 bodies.



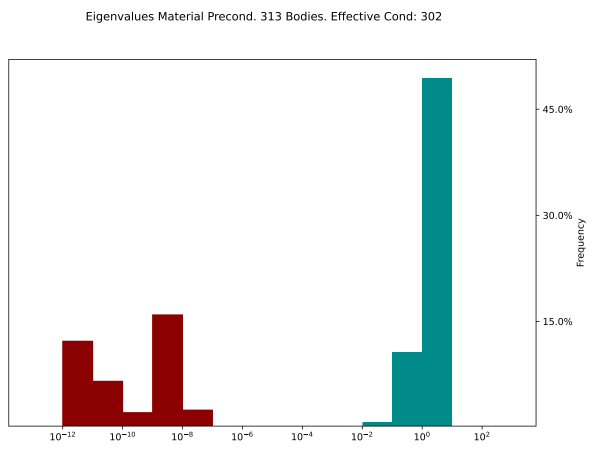
(b) Schur complement of the time step we are looking at.

Figure 20 – The scene at the timestep we are running our tests on and the Schur complement.

For the material test, we again see in Figure 21 that the eigenvalues for the original and preconditioned system have one large set of eigenvalues and the rest are small. The effective condition number for the original system is $\kappa_e(A) = 630634$ and $\kappa_e(P^{-1}A) = 302$ for the preconditioned system.



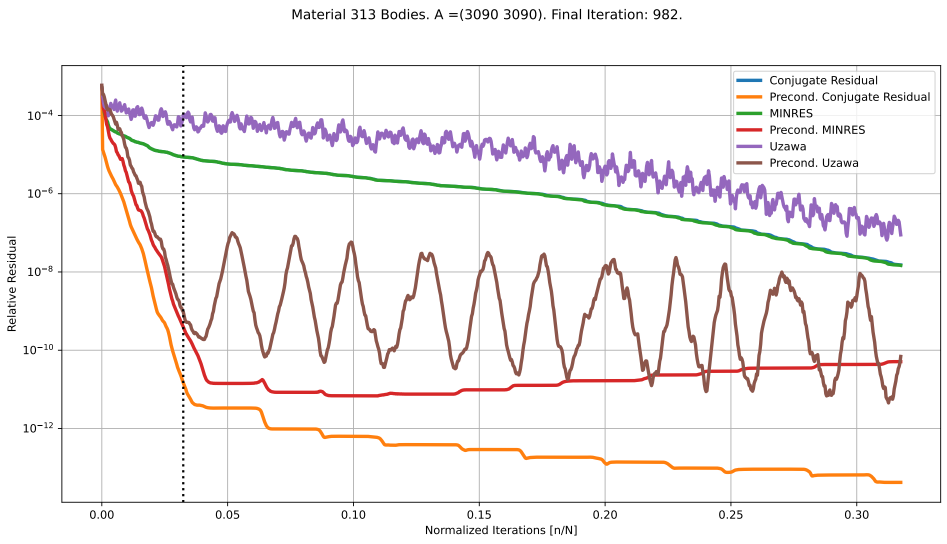
(a) Eigenvalue clustering for the material test with 313 bodies.



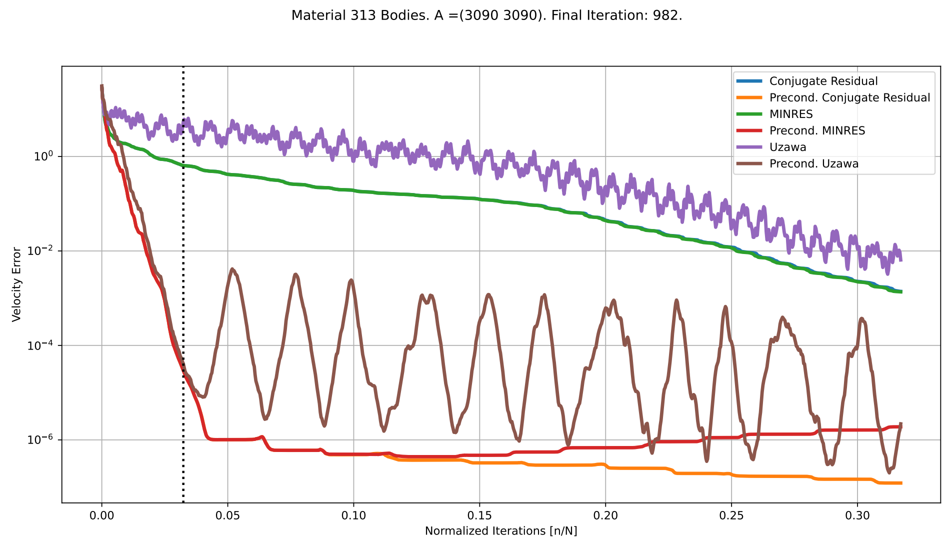
(b) Eigenvalue clustering for the preconditioner for the material test with 313 bodies.

Figure 21 – Histograms over the eigenvalue clusters for the material test with 313 bodies.

Again, the decay of the relative residual and error of the velocity looks good as seen in Figure 22. It does not perform as well the silo test. This indicates that our preconditioner has a harder time solving this type of system, which is also reflected in the effective condition number, as it is larger than the similar sized silo test.



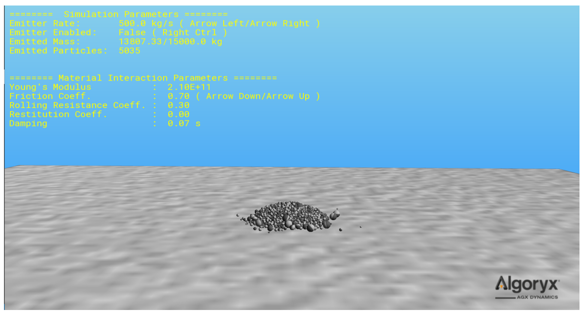
(a) Convergence plot for the relative residual for the material test with 313 bodies.



(b) Convergence plot for the error of the velocity for the material test with 313 bodies.

Figure 22 – Convergence plots for the material test with 313 bodies.

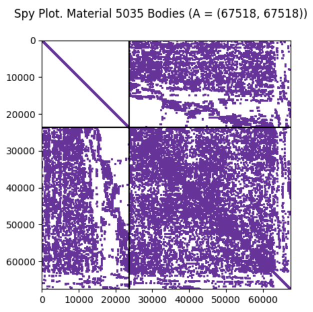
The last experment we will look at is the material test for 5035 bodies as seen in Figure 23.



```
========  Simulation Parameters ========
Emitter Rate:       500.0 kg/s ( Arrow Left/Arrow Right )
Emitter Enabled:    False ( Right Ctrl )
Emitted Mass:       13807.33/15000.0 kg
Emitted Particles:  5035


======== Material Interaction Parameters ========
Young's Modulus              :  2.10E+11
Friction Coeff.              :  0.70 ( Arrow Down/Arrow Up )
Rolling Resistance Coeff.    :  0.30
Restitution Coeff.           :  0.00
Damping                      :  0.07 s
```

(a) Material test with 5035 bodies.

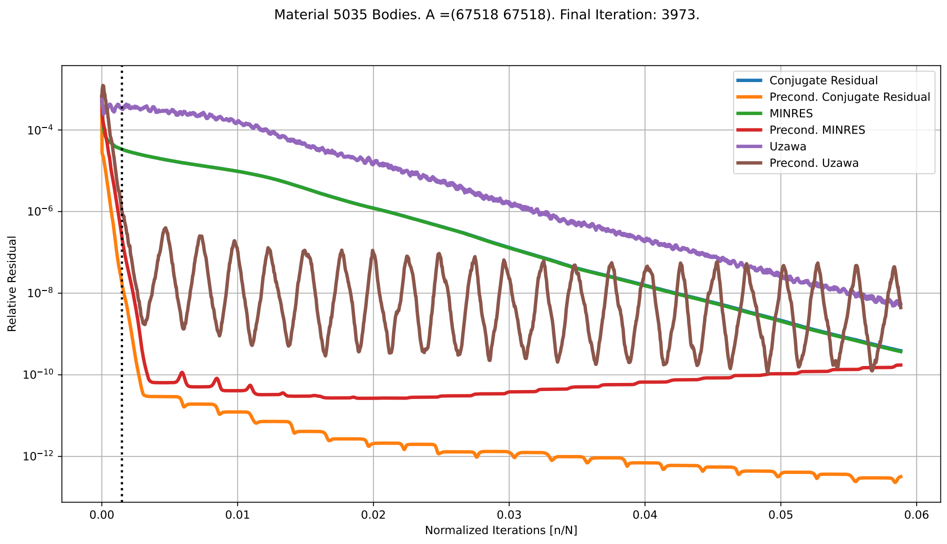


(b) Schur complement of the time step we are looking at.

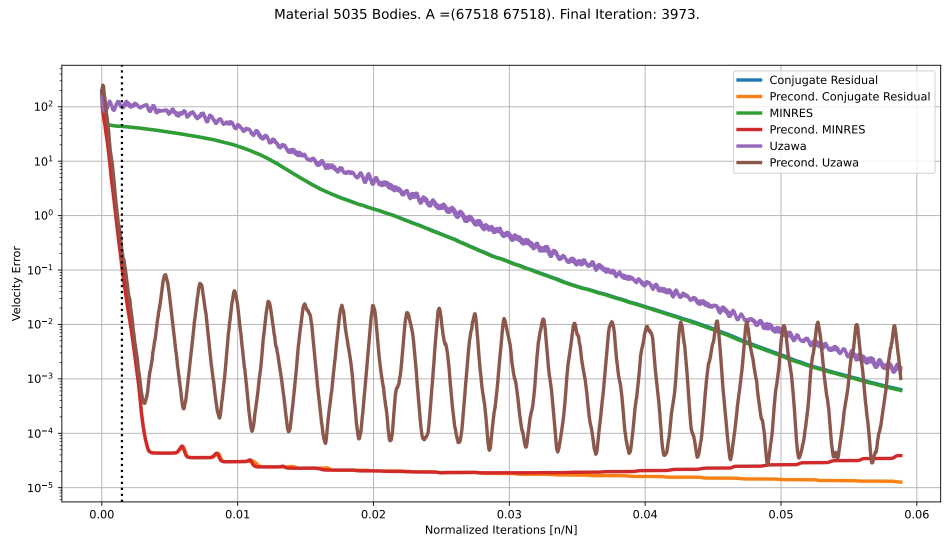
Figure 23 – The scene at the timestep we are running our tests on and the Schur complement.

For this experiment we will not look at the eigenvalues, for the same reason as for the silo test with 4988 particles, we simply can not compute them all on the available workstation.

As we can see in Figure 24, we continue to see the same behaviour with a rapid early decrease in the relative residual and error of the velocity, followed by stagnation or oscillation.



(a) Convergence plot for the relative residual for the material test with 5035 bodies.



(b) Convergence plot for the error of the velocity for the material test with 5035 bodies.

Figure 24 – Convergence plots for the material test with 5035 bodies.

The rest of the result plots can be found in Appendix A. They will not be listed here since they are all similar to the tests we have looked at in terms of the behaviour of the convergence. The rest of the results are compiled in Table 1 and 2.

Test	Bodies	Method	r_r at 100 Iterations	v_e at 100 iterations
Silo	346	Uzawa	10^{-10}	10^{-5}
Silo	346	CR	10^{-12}	10^{-5}
Silo	346	MINRES	10^{-11}	10^{-5}
Silo	668	Uzawa	10^{-8}	10^{-3}
Silo	668	CR	10^{-10}	10^{-3}
Silo	668	MINRES	10^{-8}	10^{-3}
Silo	1377	Uzawa	10^{-8}	10^{-2}
Silo	1377	CR	10^{-10}	10^{-3}
Silo	1377	MINRES	10^{-8}	10^{-3}
Silo	2567	Uzawa	10^{-6}	10^{0}
Silo	2567	CR	10^{-8}	10^{0}
Silo	2567	MINRES	10^{-6}	10^{0}
Silo	4988	Uzawa	10^{-7}	10^{-2}
Silo	4988	CR	10^{-9}	10^{-2}
Silo	4988	MINRES	10^{-8}	10^{-2}
Material	313	Uzawa	10^{-10}	10^{-5}
Material	313	CR	10^{-11}	10^{-5}
Material	313	MINRES	10^{-10}	10^{-5}
Material	631	Uzawa	10^{-8}	10^{-3}
Material	631	CR	10^{-9}	10^{-3}
Material	631	MINRES	10^{-8}	10^{-3}
Material	1271	Uzawa	10^{-8}	10^{-3}
Material	1271	CR	10^{-9}	10^{-3}
Material	1271	MINRES	10^{-8}	10^{-3}
Material	2514	Uzawa	10^{-6}	10^{-1}
Material	2514	CR	10^{-8}	10^{-1}
Material	2514	MINRES	10^{-7}	10^{-1}
Material	5035	Uzawa	10^{-6}	10^{-1}
Material	5035	CR	10^{-8}	10^{-1}
Material	5035	MINRES	10^{-7}	10^{-1}

Table 1 – The result of the preconditioner for all performed tests. The colored rows indicates the tests previously shown in this section while the white rows indicates tests that are shown in Appendix A.

Test	Bodies	Preconditioned	κ_e
Silo	346	No	72906
Silo	668	No	123656
Silo	1377	No	141024
Silo	2567	No	228939
Silo	4988	No	X
Silo	346	Yes	159
Silo	668	Yes	466
Silo	1377	Yes	580
Silo	2567	Yes	5236
Silo	4988	Yes	X
Material	313	No	630634
Material	631	No	746710
Material	1271	No	910868
Material	2514	No	790574
Material	5035	No	X
Material	313	Yes	302
Material	631	Yes	1000
Material	1271	Yes	499
Material	2514	Yes	994
Material	5035	Yes	X

Table 2 – Effective condition number for all performed tests. We do not compute the eigenvalues for the largest tests because too much memory was required to be allocated to do this, thus we can not compute the effective condition number. It is possible to compute eigenvalues with sparse matrices but then you can compute at most $N - 1$ eigenvalues, where N is the size of the system. The colored rows indicates the tests previously in this section.

7 Discussion

We can immediately see from the results that the preconditioner does make the linear systems easier to solve in the early iterations. The iterative methods tested can reach relative residuals and velocity errors many orders of magnitude smaller with the fibre preconditioner. With the fibre preconditioner there is a drastic decrease of the relative residual and the error of the velocity in the early iterations and then it starts to stagnate or oscillate. One thing that is not entirely clear is why the error of the velocity is not monotonically decreasing for any of the methods. Krylov subspace methods should get closer to the solution for every iteration, but for ill-conditioned problems this is not necessarily true. For example, the search directions of CG can lose A-orthogonality due to accumulated floating point roundoff error [11]. This could be one explanation for why we see this unpredictable behaviour.

While the convergence of the preconditioner is good, there is one thing we have deliberately not mentioned until this late in the report and that is that we have not been bothered to measure how effective it is to create the preconditioner. This has not been done because as of now, there is no point in doing it, because the construction of the preconditioner is incredibly slow. This process should be done in milliseconds, but can take up to minutes. While this makes the preconditioner unusable in its current state for any real application, there is a simple explanation for this, and the fix is not hard. As of now, the preconditioner is created by reordering the contact Jacobian G, however, as mentioned earlier in the report in a real setting one would not create this matrix. Further improvements needs to be implemented to create the preconditioner without creating the contact Jacobian. By doing this, there is a lot of performance to gain.

7.1 Future Work

There is quite a bunch of work that can be continued with for this project. First and foremost relabeling the particles and contacts instead of permuting the contact Jacobian is something that is very important. This will push down the time to create the preconditioner by a large amount. Furthermore, kernels should be implemented to be able to do all calculations without forming any matrices at all.

There is also work that can be done on how we construct the fibers. Different weights can be used for the edges instead of the weights that I used, and that can change the performance of the preconditioner. There is also something called *multipreconditioning* which as the name suggests, makes use of multiple preconditioners. This can be an interesting area to look at. We can also look at scaling the perturbation matrix T.

Finally, actually getting the fibre preconditioning into simulation is something that should be done. All the tests done during this project has just been to gauge the fibre preconditioner. Instead of just gauging it, it should be tested in a practical setting. This is very important to see that it does work as expected.

8 Acknowledgments

I would like to express my deepest appreciation for my advisor at Algoryx, Claude Lacoursière, without him this project would not have happened. He proposed the initial fibre idea, as well as providing a lot of valuable help throughout the project and feedback on the report. I would also like to thank Lars Karlsson for guidance and feedback on the report. Finally, I would like to thank Mattias Linde for valuable suggestions and feedback during the project.

References

[1] B. Andreotti, Y. Forterre, and O. Pouliquen. *Granular Media : Between Fluid and Solid.* Cambridge University Press, 2013. ISBN: 978-1-107-03479-2.

[2] D. Wang. "Accelerated granular matter simulation". PhD thesis. Umeå, Sweden: Umeå University, 2015.

[3] P. A. Cundall and O. D. L. Strack. "A discrete numerical model for granular assemblies". In: *Géotechnique* 29 (1979), pp. 47–65.

[4] M. Servin et al. "Examining the smooth and nonsmooth discrete element approaches to granular matter." In: *International Journal for Numerical Methods in Engineering* 97 (2014), pp. 878–902.

[5] A. Pyzara, B. Bylina, and J. Bylina. "The influence of a matrix condition number on iterative methods' convergence." In: *Proceedings of the Federated Conference on Computer Science and Information Systems* (Szczecin, Poland). FedCSIS, 2011, pp. 459–464. ISBN: 978-83-60810-34-4.

[6] C. Lacoursière. *Using Gauss-Seidel for Multibody Problems.* Tech. rep. 2018.

[7] Fuzhen Zhang. *The Schur Complement and Its Applications.* Numerical Methods and Algorithms. Boston, MA: Springer, 2005. ISBN: 978-0-387-24273-6.

[8] M. Molavi-Arabshahi. "Studying the effect of preconditioner scheme on the solution of convection diffusion equations". In: *Asian Journal of Scientific Research* 7.3 (2014), pp. 366–375.

[9] C. Lacoursière. "Ghost and Machines: Regularized Variational Methods for Interactive Simulations of Multibodies with Dry Frictional Contacts." PhD thesis. Umeå, Sweden: Umeå University, 2007.

[10] Y. Saad. *Iterative methods for sparse linear systems.* Society for Industrial and Applied Mathematics, 2003. ISBN: 978-0-89871-534-7.

[11] J. Shewchuk. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain.* Tech. rep. 1994.

[12] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* Johns Hopkins Studies in the Mathematical Sciences. 1996.

[13] J. R. Fernandez et al. "Preconditioning for Sparse Linear Systems atthe Dawn of the 21st Century: History, CurrentDevelopments, and Future Perspectives". In: *ISRN Applied Mathematics* 29 (2012), pp. 1–49.

[14] M. R. Hestenes and E. L. Stiefel. "Methods of conjugate gradients for solving linear systems". In: *Journal of Research of the National Bureau of Standards* 49 (1952), pp. 409–436.

[15] Dietrich Braess. *Finite Elements: Theory, Fast Solvers, and Applications in Solid Mechanics.* 1997. ISBN: 0-521-58834-0.

[16] D. Chin, L. Fong, and M. Saunders. "CG Versus MINRES: An Empirical Comparison". In: *Sultan Qaboos University Journal for Science [SQUJS]* 17 (2012), pp. 44–62.

[17] Mattias Linde. personal communication. Aug. 2021.

[18] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python.* 2001–. URL: http://www.scipy.org/.

[19] R.M. Barrett et al. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods.* Society for Industrial and Applied Mathematics, 1994. DOI: 10.1137/1.9781611971538.

[20] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[21] E. Anderson et al. *LAPACK Users' Guide.* Third. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999. ISBN: 0-89871-447-8 (paperback).

[22] A. Forsgren. *ON THE BEHAVIOR OF THE CONJUGATE-GRADIENT METHOD ON ILL-CONDITIONED PROBLEMS.* Tech. rep. 2006.