Survey of Quadratic Programming Solvers for Applying the Unilateral Incompressibility Constraint in Pressure Solving

Abstract

In a continuum approach to the simulation of granular materials self intersection between particles becomes an issue. Even with a solution to incompressible flow, the incompressibility constraint won't prevent particles from intersecting each other. The unilateral incompressibility constraint [1] provides a constraint to prevent most of these self-intersections that are visually obvious and can significantly reduce visual efficacy of a simulation. This article will explore the performance of a variety of solvers to solve the linear problems generated by my choice of discretization.

1 Introduction

The objective of this course project is to pick out a Quadratic Programming (QP) solver to use for my implementation of [2], a continuum approach to the simulation of dry granular materials using a staggered grid finite difference scheme. In the implementation of the their method an QP solver is required twice: once for the application of the Unilateral Incompressibility Condition (UIC) in the solution to the pressure of the system as a Linear Complimentary Problem (LCP), which is equivalent to a QP problem, and once in the solution of frictional stress the minimizer to a quadratic energy functional. These two formulations generate QP problems with similar stencils so I deemed it only necessary to test on the former stencil, which is also described in more depth in [1].

2 Background

First I will discuss some basic notations of constrained optimization. Then I will overview the formulations of LCP/QP problems. Finally I will describe basics required to understand the LCP/QP instances that I am intending to solve, though not in much depth because the physical phenomena are not the focus of this document.

2.0.1 Constrained Optimization

Constrained optimization is the task of minimizing some function $f : \mathbb{R}^n \mapsto \mathbb{R}$ under equality and inequality constraints. Let us notate these constraints all as c_i where $i \in \mathcal{E}$ are equality constraints and $i \in \mathcal{I}$ are inequality constraints.

The feasible set, the set of legitimate values for x in this constrained space, is defined as

$$\Omega = \{x | c_i(x) = 0, i \in \mathcal{E}, c_i(x) \geq 0, i \in \mathcal{I}\}$$

The boundary of Ω is quite important so there's a fair bit of terminology around it. The active set $\mathcal{A}(x)$ is the set of constraints for which $c_i(x) = 0$ and $\mathcal{F}(x)$, the inactive set, denotes the complement of $\mathcal{A}(x)$. The set of constraints that are active becomes a fundamental tool in the algorithms to come.

The Karush-Kuhn-Tucker (KKT) conditions are first order necessary conditions for a local minimum for an optimization problem with equality and inequality constraints and can be seen as a generalization of Lagrange multipliers to support inequality constraints. They can be seen as the system

$$\begin{aligned} \nabla_x \mathcal{L}(x, \lambda) &= 0 \\ c_i(x) &= 0, \ \forall i \in \mathcal{E} \\ c_i(x) &\geq 0, \ \forall i \in \mathcal{I} \\ \lambda_i &\geq 0, \ \forall i \in \mathcal{I} \\ \lambda_i c_i(x) &\geq 0, \ \forall i \in \mathcal{E} \cup \mathcal{I} \end{aligned}$$

where ∇_x is the gradient with respect to x and

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x).$$

2.1 Linear Complementary Problems and Quadratic Programming

2.1.1 Problem Statement

In my exploration of LCP solvers two distinct formulations of the problem definition that were used to develop algorithms. The standard formulation of linear complimentary problems is given by

$$\begin{aligned} w &= Mz + q && (1)\\ w &\geq 0 && (2)\\ z &\geq 0 && (3)\\ w^T z &= 0 && (4) \end{aligned}$$

where M usually assumed to be symmetric positive definite. The SPD assumption exists because solutions are only known if it is SPD. The reason for this assumption can be understood through the connection between LCP and quadratic programming. Any LCP instance can be seen a the solution to the Karush-Kunn-Tucker (KKT) constraints of the quadratic problem instance

$$\min_u \phi(z) = \frac{1}{2} z^T M z + q^T z \tag{5}$$

given inequality constraints that $z_i \geq 0$ for all i. If we allow for equality constraints we obtain the mixed LCP (mLCP) formulation of

$$\begin{aligned} Au + Cv + a &= 0 && (6)\\ C^T u + Bv + b &= w && (7)\\ v^T w &\geq 0 && (8)\\ v, w &\geq 0 && (9)\\ & && (10) \end{aligned}$$

which solves a slightly different quadratic system:

$$\min_z \phi(z) = \frac{1}{2} z^T Q z + r^T z \tag{11}$$

where $v \geq 0$ and

$$Q = \begin{bmatrix} A & C \\ C^T & B \end{bmatrix} z = \begin{bmatrix} u \\ v \end{bmatrix}, \quad r = \begin{bmatrix} a \\ b \end{bmatrix}.$$

The equivalence of the two formulations is rather easy to formulate. Given an LCP (M, q) we can create an equivalent mLCP by setting (A, B, C, a, b) to be $(0, M, 0, 0, q)$.

An mLCP can be reduced to an LCP problem by rearranging terms in the mLCP formulation by solving for u and substituting,which provides us with:

$$\begin{aligned} M &= B - C^T A^{-1} C && (12)\\ q &= = b - C^T A^{-1} a && (13) \end{aligned}$$

.

2.1.2 LCP and Karush-Kuhn-Tucker Conditions

Though I've mentioned that LCP problems can be written as the solutions to a quadratic programming instance, it can also be shown that the KKT first order conditions for a quadratic programming instance can easily be described

in terms of an LCP problem. The natural instance of a quadratic programming problem is a quadratic functional bounded by linear inequality constraints like this:

$$\min_x \quad \frac{1}{2}x^T Gx + x^T c \tag{14}$$
$$Ax \geq b \tag{15}$$
$$\tag{16}$$

Under the standard LCP formulation the only bound on x is positive and other constraints are not obviously represented. By applying the KKT conditions on this system, we see that a solution to the above quadratic programming instance must be a solution to the LCP given by the system below where a slack variable λ must be introduced:

$$\begin{bmatrix} G & -A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} -c \\ b \end{bmatrix}.$$

Given that if the above system has full rank and G is positive symmetric definite the solution is unique and so solving the LCP problem is equivalent to solving the quadratic problem. Therefore we see that LCP is equivalent to QP.

2.1.3 Complimentary and Minimization Duality

Maintaining the complimentary condition in an iterative process is quite difficult. When the constraint variables are strictly positive instead of directly forcing the complimentary condition in iterative algorithms, it's simpler to treat the constraint as a minimization problem. That is, two vectors $u, v \geq 0$ are complimentary to each other if and only if for any index i, $\min(u_i, v_i) = 0$ so we can reformulate our complimentary equality constraint as a minimization constraint on the min of the two vectors coefficient-wise. This will be used to determine convergence tests in the iterative algorithms that follow.

2.2 Continuum Approach to Granular Materials

A continuum treatment of granular materials allows us to simulate large bodies of granular particles without worrying about the dynamics of individual particles, which is costly. However by solving for the system as a continuum we lose small scale detail in some way. I chose to use a finite difference approach, which meant that the small scale details lost are those within individual the Eulerian grid cells, which serve as the fundamental units behind the system.

2.2.1 Discretization Scheme

Staggered grid approaches like the MAC scheme [3] are a mathematically satisfying storage method for finite difference schemes for fluid simulation. By storing physical quantities like velocity in faces to represent flux and pressure in the cell centers to represent the pressure of the volume of the cell rather than just at cell centers we can easily represent these physical quantities as differential forms on the domain, which allows us to represent our system through a discretization of exterior calculus. For instance the divergence, a 2-form (or 3-form in 3D), can be stored at a cell center and computed by the exterior derivative of the fluxes leaving the cell, which is differentiation on 1-forms (2-forms in 3D).

I followed the scheme described in [4], where I store solid boundaries are stored through signed distance functions at vertices and use the Laplace operator that he derives from the solid boundary conditions which at a very high level limits the flux over an face by the amount of blockage covering the face.

2.2.2 Physical Behavior of Sand

The rheological behavior of granular materials at a sufficiently large scale is that of a time independent viscous non-Newtonian fluid, in particular they're shear thickening. Because of its time independence the flow of a granular material can be locally defined as an energy minimization based on its stress tensor σ which we separate into the isotropic mean stress term (pressure) p and the traceless deviatoric stress term (frictional) s:

$$\sigma = -pI + s$$

where I is the identity matrix.

The shear thickening component can be expressed through a variety of yield criterion that describes the boundary constraint on the stress tensor before the granular material acts in a rigid regime. Here we use the Drucker-Prager yield condition, a continuous variant of the Mohr-Coulomb yield condition:

$$\|s\| \leq \sqrt{3}\alpha p + c$$

for some norm $\|\cdot\|$ (we'll use the Frobenius norm) and α, c being material parameters. c corresponds to cohesion, which I will assume is 0 to simulate dry granular materials.

The method I'm looking at alternates between solving for a velocity that satisfies pressure constraints and frictional constraints using a variational principle.

2.2.3 Pressure

In the above framework the static representation of granular material is stored precisely as that of a viscous fluid. The only difference represented is the yield condition that appears in the resolution of the stress. However, we're really treating granular materials as a viscous fluid we lose the fact that the granular material is composed of individual grains that cannot be compressed past some maximal density. Therefore we must enforce the UIC, which represents the maximal density obtained when packing spheres. The condition explicitly sets a maximal density

$$\rho_{max} = 2\alpha/(\sqrt{3}d_{\min}^2)$$

for a parameter α representing imperfections in packing and $d_{\min}$ representing the minimal distance allowed between objects (the radius of a particle of sand). When the density of a grid cell is below the packing density there are no pressure forces so it follows the velocity prescribed by the other forces in the system.

Within a homogeneous medium and given a density constraint, the mass of material in a finite volume can be identified with a volume fraction ϕ and the momentum of that mass can be identified with ϕv. In [2] the advection of density through pressure is given by

$$\phi^{n+1} = \phi^{n+1}|_{p=0} - \frac{\Delta t^2}{\rho\max}\nabla^2 p$$

where $\phi^{n+1}|_{p=0}$ is the density at the next time step ignoring pressure. With the addition of a complimentary constraint $p(1-\phi^{n+1}) = 0$ which says that when we have pressure forces if and only if the system is completely packed in that cell, we have an LCP problem:

$$\begin{aligned} A_1 p + b_1 &\geq 0 && (17)\\ p &\geq 0 && (18)\\ p^T(A_1 p + b_1) &= 0 && (19) \end{aligned}$$

where

$$\begin{aligned} A_1 &= \frac{\Delta t^2}{\rho\max} D_1^T D_1 && (20)\\ b_1 &= 1 - \phi^{n+1}|_{p=0} && (21) \end{aligned}$$

given that D_1 is the finite difference gradient operator on the density.

This method is sufficient for guaranteeing that particles don't usually get too close but it isn't an exact solution to having no self intersection. Therefore, after each UIC solve the simulation usually pushes each particle around in a small random walk, which works because the particles are already mostly separated out and only need to be nudged slightly.

2.2.4 Friction

Frictional energy is represented by a modified functional in this system:

$$E = \frac{1}{2}\int w\rho^n \|v\|^2 dV$$

which introduces a bit of error but improves the conditioning of the problem by reducing the variation of coefficients used by low-density portions of the discretized system. We approximate the constraints of the Druker-Prager yield

criterion with a series of planar constraints by bounding the coefficients of the tensor by $s_{\max} = \alpha p$. Minimizing this energy functional is a quadratic so we obtain the following quadratic system:

$$\begin{aligned} s^T A_2 s + b_2^T s &\geq 0 && (22)\\ s_{\max} \geq s &\geq -s_{\max} && (23)\\ & && (24) \end{aligned}$$

where

$$A_2 = \frac{\Delta t^2}{\rho_{\max}} D_2^T D_2 \quad (25)$$

$$b_2 = \frac{\Delta t}{\rho_{\max}} D_2^T \rho^n v|_{p=0} \quad (26)$$

given that s is a vector of the stress tensor coefficients and D_2 is the finite difference gradient operator on the stress tensor. The linear constraints built into this system are easily reformulated as linear constraints through the Karush-Kuhn-Tucker conditions.

2.3 Classification of Systems

The various methods I will discuss later only work on certain classes of matrices like being Positive Symmetric Semidefinite or diagonally dominant so I'll now explain why the methods I describe will apply to the systems I need to solve.

The systems that I am dealing with are represented in a staggered Eulerian grid and the matrices both represent Laplacians of different types of functions (in pressure we're dealing with a scalar function Laplacian while in friction we're dealing with a vector function Laplacian). Because of the regularity of the grids we can derive that the matrices will be composed by a constant number of off-diagonal bands. Also, because these Laplacian matrices are produced by $G^T G$ for their respective finite difference gradient matrices they must be symmetric positive semidefinite because for any x, $xG^T Gx = \|Gx\|^2 \geq 0$. They therefore have Cholesky factorizations and because of their sparsity these matrices are well represented by incomplete Cholesky factorizations.

By the structure of Laplacians we see that the sum of the values in any row must be zero and at the same time the only positive entry is the diagonal, so the matrix must be diagonally dominant.

3 Numerical Methods

For this project I looked at many algorithms for solving LCP problems and decided upon a few that I thought looked particularly promising. The most famous algorithm for solving LCP problems is Lemke's algorithm which is a complimentary pivoting algorithm. I also looked at a class of methods derived from applying linear methods to quadratic settings by projecting their solutions back into Ω after each iteration, which is not always as direct as one might imagine, but there has been significant progress in these techniques in recent years[5][6].

3.1 Convergence Criterion

Having a good stop criterion for LCP and mLCP problems can be tricky because of the complimentary condition. For LCP residuals are also not useful because the constraints will usually force the solution to be far away from where the residual goes to 0. A particularly nice and easy error metric for a normal LCP system is given by

$$E = \sum_i \left(|z_i w_i| - \min(z_i, 0) - \min(w_i, 0)\right)$$

where our current solution z and residual w are punished for negative values and also for being far from complimentary.

For mLCP we have a more sophisticated stop condition because we need to confirm that the residuals $\rho_a^k, \rho_b^k, \rho_c^k$ go to 0 where they are defined by

$$\begin{aligned} \rho_a^k &= \|Au^k + Cv^k + a\| \\ \rho_b^k &= \min_i\{v_i, (C^T u^k + Bv^k - w^k + b)_i\} \\ \rho_c^k &= \max_i\{0, -(C^T u^k + Bv^k - w^k + b)_i\} \end{aligned}$$

This leads to an error function

$$E = \max\left(\frac{\rho_a^k}{1+\|a\|}\frac{\rho_b^k}{1+\|b\|}\frac{\rho_c^k}{1+\|b\|^2}\right)$$

In the algorithms that follow I'll write "Stop Criteria" to refer to the error metrics listed above as becoming too small.

3.2 Lemke's Algorithm

Lemke's Algorithm was one of the first algorithms for solving LCP problems and intuitively what it does is add an extra variable to the system which generates a trivial solution to the whole LCP system and then works on trying to minimize the weight of the added variable without breaking the LCP system until the variable disappears. It does so by adding the variable d and coefficient z_0 to the standard LCP system and performing Gauss-Jordan pivots on the system like the simplex algorithm:

$$\begin{aligned} w &= Mz + q + dz_0 && (27)\\ w &\geq 0 && (28)\\ z &\geq 0 && (29)\\ d &> 0 && (30)\\ w^T z &= 0 && (31) \end{aligned}$$

This system is usually represented through a tableau:

	$w_1 ... w_n$	$z_1 ... z_n$	z_0	
w	I	M	d	q

Following the discussion found in [7] we represent the set of points that can be represented by the span n column vectors of $[I|M]$ with positive coefficients is a positive cone. The LCP problem can be thought of as the discovery of which complimentary cone q lies in. Finding the right complimentary cone is equivalent to finding which dimensions should be active and which ones shouldn't, as that determines the n column vectors that define the complimentary cone q lies in. It's pretty obvious that for any $d > 0$ we can set some sufficiently large z_0 such that $q + dz_0$ belongs in the complimentary cone of I. Lemke's algorithm starts off in that cone and wanders around the other cones using minimizing the number of nonzero entries in d as a measure of how close the system is to the right complimentary cone.

The algorithm is set in two phases: one to initialize a cone that uses a nonzero z and nonzero w entries only and one to iterate through cones until it reaches one that doesn't include our added extra d in its basis.

The algorithm starts by trying to push d into the basis by switching out some w_r. This is done by finding the largest$\bar{z_0}$ that will keep w nonnegative, which will set at least one w_r to be zero and remove it from the basis. From there we pivot w_r on with z_0 and put z_r as our driving variable. With z_0 in the system there are a bunch of variables z_i and w_i for which precisely one of them is in the active basis and one pair where neither is in the active basis. One of these should always be the driving variable.

In the second phase Lemke's algorithm iterates over deciding which dimensions should be entering and leaving the basis and pivoting the two. The dimension entering is the driving and the variable leaving is the argmin$\{\frac{q_i}{c_i}\}$ where c is the current driving variable. From there we do a pivot and if we see that z_0 is the driving variable we are done.

In this algorithm let y_r be the driving variable, b_s be the sth column of the current basis, and d the columns of the driving variable.

3.3 Projection Methods

Without constraints, solutions to LCP problems are the solutions to linear systems of equations. Because of that it's fairly natural to consider using iterative linear solvers that tweak their solutions when constraints become active, and that's what projection methods do. The naive method to produce projection methods is to perform one iteration of a given method and then the project the current solution to the feasible set, but there's a technique called subspace minimization that can improve the performance of most projection methods by eliminating variables that are not necessary.

Algorithm 1 Lemke's Algorithm

```
if q ≥ 0 then
    x ← 0
end if
d ← d^0 ← [1111...1]^T
r ← argmin q_i/d_i
Pivot < z_0, w_r >
y_r ← z_r
while z_0 is not charging do
    s ← argmin q_i/d_i
    Pivot < y_r, b_s >
    Set y_r to be the complement of b_s
    Set d to be the col y_r
end while
```

3.3.1 Subspace Minimization

If we assume that active constraints tends to stay active we can remove the dimensions corresponding to active set to temporarily reduce the burden of doing computations in high dimensional spaces. In LCP the active set corresponds with variables x_i being set to 0 and in mLCP problems this corresponds to setting u_i to be 0 so it's fairly easy to detect. From there one reduces the system by removing the columns and rows corresponding to active set constraints. For an LCP we get a reduced system

$$\hat{M} = PMP^T \quad \hat{q} = Pq$$

and for an mLCP we get

$$\hat{C} = CP^T \quad \hat{B} = PBP^T \quad \hat{b} = Pb \quad \hat{v} = Pv$$

where P is a projective map removing active set elements. If we assume the first m variables are in the active set we get that $P = [0|I_{n_b - m}]$ where n_b is the number of dimension of b.

In general the stability of the active set is not guaranteed but for some algorithms it seems to be a good idea. If the active set is somewhat stable we can still alternate between solving the full system and solving reduced systems to improve performance, but amazingly in some cases we can continue reducing the system without ever looking back at the full system. In those cases the dimensionality of the problem becomes shrinks as we get closer to the solution and the speed improvement from subspace minimization can be incredibly powerful.

For most iterative algorithms we can apply the following:

Algorithm 2 Subspace Minimization

```
while Stop Criteria do
    Run several steps of algorithm on full system
    Run several steps of algorithm on reduced system
end while
```

3.3.2 Gradient Projection

An obvious algorithm to try is probably gradient descent, which has a constantly decreasing solution and guaranteed convergence for problems well beyond linear systems of equations. For a standard LCP problem one can move in the direction $-g$ where $g = Mx + q$ which is the gradient of the quadratic

$$\phi(x) = \frac{1}{2} x^T M x + q^T x.$$

From this direction we need to determine where the ray cast from our current x in the gradient direction hits the boundaries of the feasible set. If we reach the Cauchy point then we can take that point, but if we hit the boundary of the feasible set we must stop traveling at that boundary. For LCP problems we can obtain this boundary by taking

$$\bar{t}_i = x_i / g_i$$

where x_i and g_i are the ith component of the vectors x and g. We can then rewrite the projected position of the ray $x(t)$ at parameter t by

$$x_i(t) = \begin{cases} x_i - tg_i & t \leq \bar{t}_i \\ x_i - \bar{t}_i g_i & \text{otherwise} \end{cases}.$$

Finding the minimum of this projected ray can be found by doing a piece by piece analysis of the objective function on each interval $[t_i, t_{i+1}]$.

This algorithm is particularly prone to a particular form of subspace minimization in that we can reduce the size of the system on the active set of the Cauchy point. Once we get to the Cauchy point we can try to find a point x^+ on the hyperplane that the Cauchy point lies on to get a bit of extra performance on each step in a heavily reduced system.

Algorithm 3 Gradient Projection with Subspace Minimization

while Stop Criteria **do**
  find the Cauchy point x^c from x_k
  Find feasible x^+ such that $\phi(x^+) \leq \phi(x^c)$
  $x^{k+1} \leftarrow x^+$
end while

The idea of taking an iterative technique and projecting it onto the constraint boundary is a fairly fundamental idea and is the basis of most techniques extending linear techniques into quadratic ones.

3.3.3 Projected Gauss-Seidel

Derived from the same principles as the original Gauss-Seidel algorithm, the Projected Gauss Seidel algorithm seems to be one of the primary choices for solving LCP problems in rigid body simulations due to its ability to converge on a fairly wide set of matrices expediently and with constant memory overhead. Convergence can be shown for positive definite matrices and diagonally dominant matrices.

For LCP problems this can be derived by separating the matrix M into

$$M = L + D + U$$

where L and U are strictly lower and strictly upper triangular matrices and D is diagonal. This allows for the separation of our system to $(L + D)x = -q - Ux$ which we can be separated into iterative steps $(L + D)x^{k+1} = -q - Ux_k$, which gives us $x^{k+1} = (L + D)^{-1}(Ux_k + q)$. By taking advantage of the lower triangular $L + D$ we can write the updates for each element in x^{k+1} as

$$\begin{aligned} x_i^{k+1} &= -\frac{1}{M_{ii}}\left(q_i + \sum_{j<i} M_{ij}x_j^{k+1} + \sum_{j>i} M_{ij}x_j^k\right) \\ &= -\frac{1}{M_{ii}}\left(M_{ii}x_i^k - M_{ii}x_i^k + q_i + \sum_{j<i} M_{ij}x_j^{k+1} + \sum_{j>i} M_{ij}x_j^k\right) \\ &= x_i^k - \frac{1}{M_{ii}}\left(q_i + M_{ii}x_i^k + \sum_{j<i} M_{ij}x_j^{k+1} + \sum_{j>i} M_{ij}x_j^k\right). \end{aligned}$$

The reason for the last two manipulations is that the entries in M used are the ith row and the entries of x can be represented as the partial update of x^k to x^{k+1} when updating the values entry by entry. We can therefore represent the difference of the update of the difference as the inner product of the ith row of M and the x stored in memory, which I represent using a Matlab-esque representation $q_i + M_{i:} \cdot x$.

This algorithm naturally extends to cases where we have more sophisticated boundary cases which are more easily represented as mLCP problems. The formulation of the Projected Gauss-Seidel for mLCP problems can be done by alternating between solutions for u in

$$Au = -Cv - a$$

and solving for v in

$$Bv = -C^T u - b.$$

This process is intuitively the application of the mLCP→LCP procedure without explicitly computing the inverse of A or doing the matrix-matrix multiplications.

Algorithm 4 Projected Gauss-Seidel

$x \leftarrow q$
while Stop Criteria **do**
  for $i = 1...N$ **do**
    $\delta x_i \leftarrow q_i + M_{i:} \cdot x$
    $x_i \leftarrow \max(0, x_i - \delta x_i)$
  end for
end while

Algorithm 5 Projected Gauss-Seidel for mLCP

$x \leftarrow q$
while Stop Criteria **do**
  for $i = 1...N$ **do**
    $\delta u_i \leftarrow a_i + A_{i:} \cdot u$
    $u_i \leftarrow u_i - \delta u_i$
  end for
  for $i = 1...N$ **do**
    $\delta v_i \leftarrow b_i + M_{i:} \cdot v + (C^T)_{i:} \cdot u$
    $v_i \leftarrow \max(0, v_i - \delta v_i)$
  end for
end while

3.3.4 Projected Jacobi

Another linear algorithm that can be amended to work with LCP problems is the traditional Jacobi method. As a linear method it's used less frequently than Gauss-Seidel because of it converges slower on a single core and requires an extra vector of memory, but its parallelizability has made its performance beat Gauss-Seidel when run on multi-core processors. This paralleilzability is not lost in the gradient projection.

Borrowing the same separation method as the linear method we see that if we take

$$M = D + R$$

where D is diagonal and R is everything else in M not included in D. Through this separation we can separate $Mx + q$ into $Dx + Rx + q$, which gives of an iterate when we set it to 0. $Dx_{k+1} = -(Rx_k + q)$. As in the previous algorithm the idea is to run it until the gradient produced by each time step reaches zero while forcing the entries of x_k to be positive. Through a slight reordering we can rewrite the above equation as

$$\begin{aligned} x_{k+1} &= -D^{-1}((R + D - D)x_k + q) \\ &= -D^{-1}(-Dx + Mx_k + q) \\ &= x - D^{-1}(Mx_k + q) \end{aligned}$$

which gives us the following algorithm:

Algorithm 6 Projected Jacobi

$x \leftarrow q$
while Stop Criteria **do**
  $\delta x \leftarrow D^{-1}(Mx + q)$
  $x \leftarrow \max(0, x - \delta x)$
end while

A similar procedure can be performed for mLCP problems with the Jacobi method through the process of swapping between solving for u and v.

3.3.5 Modified Proportioning with Reduced Gradient Projections

By applying the same generic projection technique as above one would expect at the very least convergence, but from what I can tell the naive projection of conjugate gradient does not converge. One can, however, make conjugate

gradient work by taking special efforts to take into account the state of the system with respect to the boundary of Ω.

The method I wound up looking at is a specialized implementation of conjugate gradient with a different variant of subspace minimization. Every step can be one of three different methods depending on how close the standard CG direction is to crossing a constraint boundary. In the context that I'm aware of it supports linear constraints of the form $x_i \geq l_i$. When it reaches a boundary it minimizes its search to the subspace where the constraint is always active, causing the dimensionality of the problem to decrease.

The method picks one of three step choices each iteration: a conjugate gradient step, an expansion step, or a proportioning step. The main step is the standard conjugate gradient step and the other two steps are corrections to make sure that the system is constrained enough to converge. This decision is made by determining the relative magnitude of the current descent direction that breaks through Ω through two conditions. Let g be the current descent direction and let β and ϕ be vector functions defined as follows:

$$\begin{array}{ll} \phi_i(x) = g_i(x) \forall i \in \mathcal{F}(x) & \phi_i(x) = 0 \forall i \in \mathcal{A}(x) \\ \beta_i(x) = 0 \forall i \in \mathcal{F}(x) & \beta_i(x) = \min\{g_i, 0\} \forall i \in \mathcal{A}(x). \end{array}$$

ϕ,the free gradient operator, describes the proportion of g that that is not in a constrained dimension while β, the chopped gradient operator, describes the proportion of g that goes in a constrained dimension without breaking the constraint. The KKT conditions are met when $\nu(x) = \phi(x) + \beta(x) = 0$.

The first of the two correctional steps is the expansion step, which happens when the conjugate gradient step is about to activate a constraint. With P_Ω as a projection operator back onto Ω and $\bar{\alpha}$ some minimal step size parameter set in the interval $(0, \|M^{-1}\|]$ we set the expansion step to be

$$x^{k+1} = P_\Omega(x^k - \bar{\alpha}\phi(x^k)).$$

This projection operator can be made more explicit by defining a reduced free gradient operator $\bar{\phi}$ which will set x^{k+1} to activate the constraint in the step. It's defined by:

$$\bar{\phi}(x) = \min\{(x_i - l_i)/\bar{\alpha}, \phi_i\}$$

which gives a new notation that

$$x^{k+1} = P_\Omega(x - \bar{\alpha}g(x)) = x - \bar{\alpha}(\bar{\phi}(x) + \beta(x)).$$

The proportioning step is checked before the conjugate gradient step length to push away from useless active constraints as much as possible . This is done with the inequality

$$\|\beta(x)\|^2 \leq \Gamma^2 \bar{\phi}(x)^T \phi(x)$$

with a parameter Γ in $(0, 1]$ to determine how large of a proportion of ϕ we're going to allow the gradient to face into Ω. If the proportioning step is chosen it does one round of conjugate gradient using β as the gradient for that step.

3.3.6 Cholesky Factorization for Preconditioned Modified Proportioning with Reduced Gradient Projections

In Modified Proportioning with Reduced Gradient Projections there's a choice of preconditioner which allows allows us to use a preconditioner that has proven to be incredibly powerful for solving for pressure and viscosity for normal viscous fluids. In particular the Modified Incomplete Level-0 Cholesky factorization provides excellent convergence as the preconditioner for preconditioned conjugate gradient. The choice of using a Cholesky factorization is fairly evident by the regular pattern of the stencils used in the above pressure and friction schemes: they're both composed of a sparse number of symmetric positive semidefinite matrices that are only filled on a small set of diagonals.

Since MPRGP requires directions in dynamic subspaces of Ω depending on which constraints are active, a different preconditioner is required for every permutation of active constraints. These new factorizations are fairly easy to compute and can be estimated by the following scheme. Letting $M_{\mathcal{FF}}$ be the free part of M and assuming that the free indices all sit in the front of the matrix we can find a Cholesky factorization for $M_{\mathcal{FF}}$ by substituting M with

Algorithm 7 Modified Proportioning witih Reduced Gradient Projections

Set $r \leftarrow Ax - b, p \leftarrow \phi(x)$
while $\nu(x) > \epsilon$ **do**
  if $\|\beta(x)\|^2 \leq \Gamma^2 \bar{\phi}(x)^T \phi(x)$ **then**
    $\alpha_{cg} \leftarrow r^T p / p^T A p$
    $\alpha_f \leftarrow \max\{\alpha : x - \alpha p \in \Omega\}$
    if $\alpha_{cg} \leq \alpha_f$ **then**
      Conjugate Gradient Step
      $x \leftarrow x - \alpha_{cg} p$
      $r \leftarrow \phi(x) - \alpha_{cg} A p$
    else
      Expansion Step
      $x \leftarrow x - \alpha_f p$
      $r \leftarrow r - \alpha_f A p$
      $x \leftarrow P_\Omega(x - \bar{\alpha}\phi(x))$
      $r \leftarrow Ax - b$
      $p \leftarrow \phi(x)$
    end if
  else
    Proportioning Step
    $d \leftarrow \beta(x)$
    $\alpha_{cg} \leftarrow r^T d / d^T A d$
    $x \leftarrow x - \alpha_{cg} d$
    $r \leftarrow r - \alpha_{cg} A d$
    $p \leftarrow \phi(x)$
  end if
end while

LDL^T:

$$\begin{aligned} M_{\mathcal{F}\mathcal{F}} &= \begin{bmatrix} I \\ 0 \end{bmatrix} M \begin{bmatrix} I \\ 0 \end{bmatrix}^T \\ L_{\mathcal{F}\mathcal{F}} D L_{\mathcal{F}\mathcal{F}}^T &= \begin{bmatrix} I \\ 0 \end{bmatrix} L D L^T \begin{bmatrix} I \\ 0 \end{bmatrix}^T \\ L_{\mathcal{F}\mathcal{F}} D L_{\mathcal{F}\mathcal{F}}^T &= \begin{bmatrix} I \\ 0 \end{bmatrix} L D L^T \begin{bmatrix} I \\ 0 \end{bmatrix}^T \\ L_{\mathcal{F}\mathcal{F}} &= \begin{bmatrix} I \\ 0 \end{bmatrix} L. \end{aligned}$$

This lets us solve $M_{\mathcal{F}\mathcal{F}} x = c$ with the original factorization with modified forward/backward algorithms that ignore columns/rows related to the active set.

4 Implementation Details

Most of the methods above theoretically only work for symmetric positive definite matrices but my system is in fact positive semidefinite. However, the systems I use have an obvious desirable solution from the physical nature of the system. The degeneracies in the system happen only when a grid cell has either no neighboring cells for which it can advect fluid material with due to constraints such as the cell being inside a solid. There is therefore no pressure term to solve in this system, which means that in the solution to the solve there should be a 0 in the entry corresponding to the degeneracy. The degenerate dimensions in the system are axis aligned and are signified by empty rows and columns, and in particular 0 diagonals. Therefore in all of my algorithms whenever it required a degenerate dimension I could check if the diagonal was 0 to determine whether to skip the step or not.

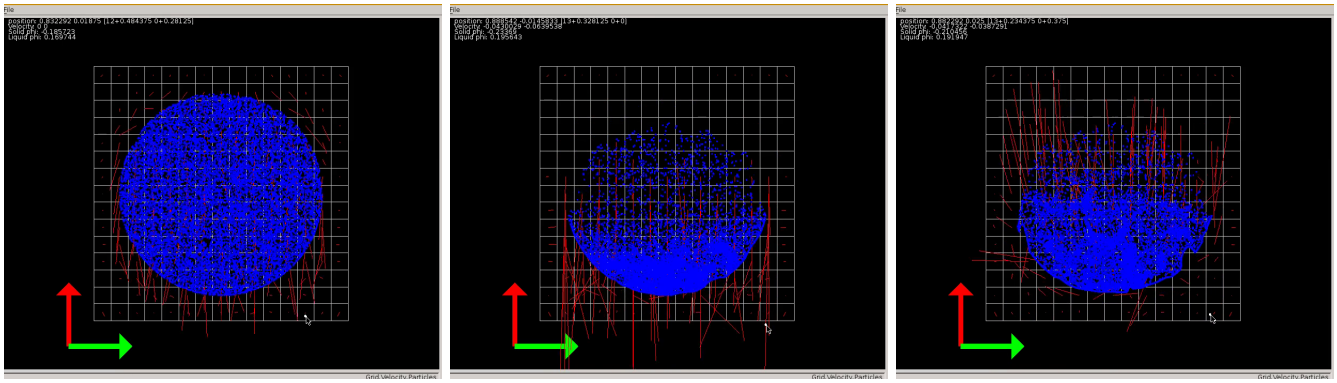

Figure 1: Images of the system at initialization, when particles are settling at the bottom, and a couple of frames after the bottom went over density constraint

4.1 Eigen

I chose to implement everything in C++ using the Eigen library to handle low level linear algebra operations and matrix/vector representations. Eigen is a templated numerical library that uses the Curiously Recurring Template Pattern to support quite a few interesting advanced features. Eigen felt like a good choice because of its impressive benchmark claims and its expressiveness seems comparable to Matlab at times. I wrote a few naive tests to compare it to some standard BLAS implementations I had and Eigen outperformed them all. However, it's still a fairly new library, only a couple of years old, and so I ran into a few issues while using it. It's still not very well documented and there's a few issues that popped up throughout the development of this project.

4.1.1 Eigen Sparse Matrices

In particular I used their sparse matrix representation quite heavily which is still quite unstable. In the current stable Eigen build 3.0.* there's two sparse matrix implementations SparseMatrix and DynamicSparseMatrix, which were sufficient for a while until I wanted to have lower level access to the matrices for generating Cholesky factorizations. At a high level the representation used an array of tuple vectors which provided $O(1)$ access to an outer dimension but $O(nz_i)$ access to an arbitrary inner element, where nz_i is the number of nonzero entries in major dimension i, which proved to be too slow compared to other solvers that I wanted to beat. Later into the semester I discovered their development branch 3.1.* builds use a completely new sparse matrix representation that is significantly better.

They merged the two sparse matrix classes into a singular one and allowed for the sort of lower level access I needed like iterators. Because I could now iterate through inner vectors in constant time I could optimize my Cholesky factorizations by a fairly visible result when running my fluid simulator which uses a preconditioned conjugate gradient solver.

5 Experiment

I implemented six different algorithms and tested for their time performance on 10000 test cases that I generated at several different resolutions: Projected Gauss-Seidel, Projected Gauss-Seidel with Subspace Minimization, Projected Jacobi, Projected Jacobi with Subspace Minimization, Modified Proportioning with Reduced Gradient Projections, and Preconditioned Modified Proportioning with Reduced Gradient Projections.

I've had success in solving linear problems with the same M matrices using preconditioned conjugate gradient, so I decided to try applying PMPRGP, which theoretically is the extension of preconditioned conjugate gradient into the LCP domain. I tried Projected Gauss-Seidel because of its popularity in the video game community and Projected Jacobi was an obvious next step after Projected Gauss-Seidel, which is the direction industrial linear solvers seem to moving in due to its high parallelizability. I set my subspace minimization algorithms to alternate between 5 rounds of full solves and 3 rounds of reduced rounds and for MPRGP I set $\Gamma = 1$ and $\bar{\alpha} = \min_{i,A_{ii}\neq 0} \|A_{ii}\|$.

5.1 Test Cases

In order to create realistic The test cases were created by a slightly modified physical simulation that did not conserve energy to maximize the number of novel configurations the system could encounter. This modified system

was created using an odd set of parameters to cause the impulses created by the UIC constraint being activated to be explosive, so when particles started to settle at the bottom of the domain the UIC constraint would activate and the particles would be thrown into the air. One issue that came up was that the unnatural impulses exacerbated the small scale discretization errors which I did not realize would cause such noticeable issues in later frames and caused material to pack together into small clumps of overly-dense particles that refused to separate. This effect was particularly noticeable the boundaries of the system. My method of manipulating the physics also made small numbers of particles invisible to the solver and they would stick in the air.

The matrices I used were chosen from a circular domain within a square grid. I did not explore higher dimensioned systems because creating the data through Projected Gauss Seidel took too long / the data sets that I managed to create of higher dimensions took too long to run within reasonable time given how slow PMRGP was taking.

5.2 Results

The results were mostly what I expected on a single thread implementation of each of the algorithms. I was running on a Intel i7-2600 3.4GHz with 8GB of memory. With the exception of MGRGP all of the other algorithms seem to have the same convergence properties, which is probably because they are all of the same variety of problem.

As expected, on a single thread Gauss-Seidel performed better than Jacobi and subspace minimization improved the performance of both algorithms to the point where on the highest resolution Jacobi with subspace minimization beat plain Projected Gauss-Seidel. MPRGP seemed to start outperforming all of the other algorithms in the highest resolution.

Preconditioning MPRGP reduced the number of iterations significantly below that of any other algorithm, but only at the highest dimension that I ran did it outperform any of the other algorithms. I suspect this is really due to the "small" dimensionality of the matrices considered and that the cost of preconditioning will be diminished for longer input sizes. This seems to have only happened on the highest resolution I used but I suspect that its performance will even beat Projected Gauss-Seidel with subspace minimization eventually.

OpenMP seemed to only hurt my results, which was a bit surprising, but that might also be due to the small resolutions I was using and my novice usage of OpenMP.

6 Conclusion

The best results were obtained by MPRGP at the resolutions I checked so I will probably use it to solve the systems I am dealing with for lower dimensions, like when prototyping configurations, but for higher resolution simulations I will probably try preconditioning it again.

References

[1] R. Narain and M. C. Lin, "Free-Flowing Granular Materials with Two-Way Solid Coupling," pp. 1–9, 2005.

[2] R. Narain, A. Golas, S. Curtis, and M. C. Lin, "Aggregate dynamics for dense crowd simulation," *ACM Transactions on Graphics*, vol. 28, p. 1, Dec. 2009.

[3] F. Harlow and J. Welch, "Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface," *Physics of fluids*, 1965.

[4] C. Batty and F. Bertails, "A fast variational framework for accurate solid-fluid coupling," *ACM Transactions on Graphics (TOG)*, 2007.

[5] J. Morales, J. Nocedal, and M. Smelyanskiy, "An algorithm for the fast solution of symmetric linear complementarity problems," *Numerische Mathematik*, vol. 111, no. 2, pp. 251–266, 2008.

[6] Z. Dostal, "Optimal quadratic programming algorithms: with applications to variational inequalities," 2009.

[7] K. Murty, "Linear complementarity, linear and nonlinear programming," 1988.

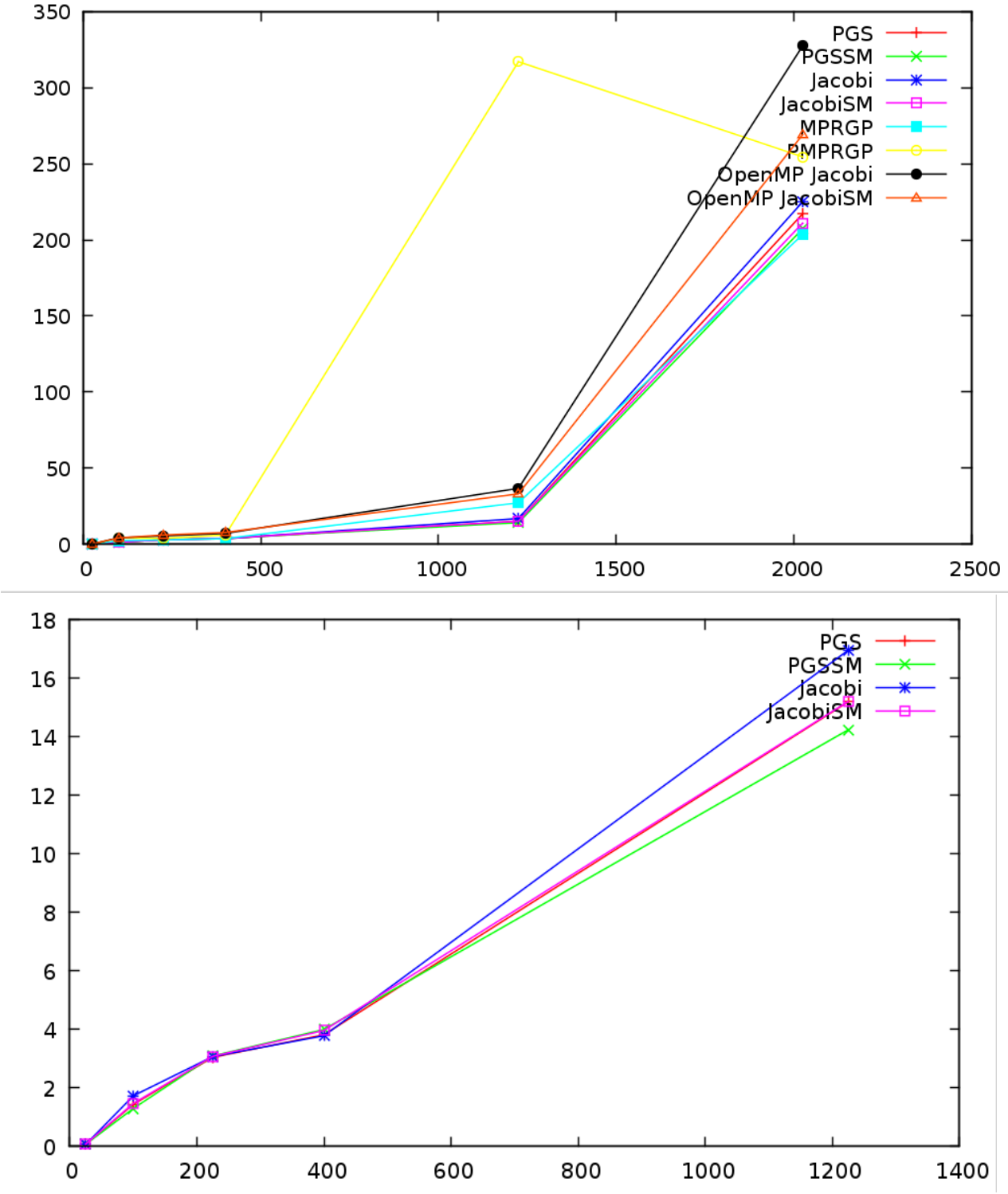

Figure 2: Top image: the performance of all of the different algorithms I ran. Bottom image: Just the two variants of Projected Gauss-Seidel and Projected Jacobi to show their relative performance more clearly. Both graphs are of the number of seconds it took to solve 10000 problems.