Department of Physics
Umeå University June 28, 2021

Predicting Particle Dynamics Using Coarse Grained Fields

Klaas von der Brelje

Advisor: Erik Wallin Examiner: Martin Servin

Contents

1 Introduction

Coarse graining is a method to compute fields from a set of particles. In this thesis we will explore this process in both a theoretical setting and a numerical simulation. We will start by providing a theoretical description and investigating some problems that arise when transitioning from an continuous to discretized fields. In particular, we will show that the total properties of the particles, such as their mass, are not conserved in the coarse graining process in a finite grid. An alternative approach is put forth, that adresses these issues.

We will then develop optimized numerical algorithms to efficiently compute the coarse grained fields in practice. Their computational complexity is analyzed and briefly compared to a naive implementation.

The algorithms are applied to a simulation of a rotating drum, filled with particles, to compute fields of various quantities. The issue of non-conservation and its resolution in the alternative approach are adressed again and demonstrated in practice.

A machine learning model, based on a neural network, is then introduced to try and predict the particle motion. The key element here is that the model works purely with data local to a given particle. Its ultimate goal is to predict the particle dynamics while avoiding the expensive computation of explicitly solving the equations of motion. The accuracy of the model is determined and possible error sources are investigated.

2 Theory

2.1 Notation Convention

Throughout this report equations will involve quantities that depend on a multitude of different values, that might lead to confusion. This section aims to establish clarity and a natural understanding of all the indices and arguments used from here on.

Particle properties will always have at least a single index that identifies the particle. This is always written as a subscript and, for variable indices, is a letter of the roman alphabet. For example the position of particle i will be written as $\mathbf{r}_i$.

The same holds true for contacts, with the addition that a contact's index i can be resolved into a pair of indices that refer to the particles involved in the contact. An example is the normal of a contact i between particle a and particle b, which would be written as $\mathbf{n}_i = \mathbf{n}_{ab}$.

While tensors are denoted with bold symbols, single components and scalars use regular letters. Any tensor, including vectors but not scalars, can appear with greek superscripts, which specify a single component of such a tensor. An example is the component μ of the position of particle i, which would be written as r_i^μ.

Fields will share some of the same symbols as the particle and contact properties but will never occur with the roman subscripts that identify a particle. Instead, the fields may include the additional spatial dependence as a parameter. For example the momentum (density) field may be written as $\mathbf{p}\left(\mathbf{r}\right)$.

Some quantities may have an additional superscript to denote entirely seperate particle properties or fields. An example are the kinetic and contact stress $\left(\boldsymbol{\sigma}^{\mathrm{k}}\right)_i \neq \left(\boldsymbol{\sigma}^{\mathrm{c}}\right)_i$. To differentiate these superscripts from tensor components they are enclosed in parentheses with the symbol of the quantity itself.

Lastly, because particle indices often occur more than once in a single term, the Einstein summation convention is not used. Summation over indices will always be explicit to avoid confusion by unintended summation.

2.2 Coarse Graining

The fundamental idea of coarse graining is to replace a set of discrete particles by continuous fields. These fields inherit the properties of the real particles, but are defined as functions of coordinates. The notion of discrete elements gets replaced by a continuum description in which the particles are no longer treated as fundamental.

While there are other methods to transition from particles to fields, coarse graining is defined as follows: To calculate the value of an arbitrary field A at position $\mathbf{r}$, we sum the weighted contributions of the corresponding particle property A_i.

$$A(\mathbf{r}) = \sum_i \phi(\mathbf{r}_i - \mathbf{r}) A_i \tag{1}$$

The weights are calculated as the scalar function ϕ acting on the difference between a given particle's position $\mathbf{r}_i$ and the position $\mathbf{r}$ at which the field is calculated. This function ϕ is called the *coarse graining function*.

Using this definition for the basic fields, the coarse graining process has been studied in great detail[3, 4, 5, 6], including the more complex derived fields, discussed in this thesis.

We can expand this definition by treating each particle individually as a field with a delta distribution, which is integrated over.

$$A(\mathbf{r}) = \sum_i \phi(\mathbf{r}_i - \mathbf{r}) A_i = \sum_i \int_{\mathbf{R}^3} d\mathbf{r}' \phi(\mathbf{r}' - \mathbf{r}) A_i \delta(\mathbf{r}' - \mathbf{r}_i) \tag{2}$$

The integral and sum commute to give a single integral over the sum of individual particle fields.

$$A(\mathbf{r}) = \int_{\mathbf{R}^3} d\mathbf{r}' \phi(\mathbf{r}' - \mathbf{r}) \sum_i A_i \delta(\mathbf{r}' - \mathbf{r}_i) \tag{3}$$

This sum can be considered a single field $A'(\mathbf{r}') = \sum_i A_i \delta(\mathbf{r}' - \mathbf{r}_i)$ in which all particles are treated as point-like.

$$A(\mathbf{r}) = \int_{\mathbf{R}^3} d\mathbf{r}' \phi(\mathbf{r}' - \mathbf{r}) A'(\mathbf{r}') \tag{4}$$

With this field of point-like particles in place, it now becomes obvious that the coarse grained field is simply a convolution of A' with the coarse graining function.

$$A(\mathbf{r}) = (\phi * A')(\mathbf{r}) \tag{5}$$

The coarse graining process can thus be generalized by allowing any distribution A'. Because the convolution (with the coarse graining functions we will look at) is akin to smoothing, we call the density A' the *microscopic density* or *true density*. The coarse graining function ϕ can also be called the *smoothing kernel* in this context.

Using a normalized function as the smoothing kernel, the integral of the coarse grained field equals the integral of the true field. This ensures that the overall properties of the system, e.g the total mass and momentum, are conserved during the coarse graining process.

$$\int_{\mathbf{R}^3} d\mathbf{r}\, A(\mathbf{r}) = \int_{\mathbf{R}^3} d\mathbf{r}\, (\phi * A')(\mathbf{r}) = \left(\int_{\mathbf{R}^3} d\mathbf{r}\, \phi(\mathbf{r})\right)\left(\int_{\mathbf{R}^3} d\mathbf{r}\, A'(\mathbf{r})\right) = \int_{\mathbf{R}^3} d\mathbf{r}\, A'(\mathbf{r}) \tag{6}$$

2.2.1 The Coarse Graining Function

The definition of the coarse graining process in equation 1 allows for any scalar function to be used as the coarse graining function ϕ. However, some additional properties are desirable. First and foremost, it should be positive semi-definite $\phi(\mathbf{r}) \geq 0$ and normalized $\int \mathrm{d}\mathbf{r}\, \phi(\mathbf{r}) = 1$. If we impose smoothness as an additional requirement, that allows us to use the derivative $\partial^\mu \phi$ to define additional fields. Also, for isotropic systems the coarse graining function should be spherically symmetric and thus only depend on the distance $\phi(|\mathbf{r}|)$.

There are plenty of functions that fullfill all these requirements, but we will look at only two of them. In both cases there will be a characteristic width w, which is called the *coarse graining scale*. This coarse graining scale can have a massive effect on the results of the entire process, as is demonstrated in figure 1.

Perhaps the simplest choice for ϕ is a radial Heaviside distribution.

$$\phi(\mathbf{r}) = \frac{3}{4\pi w^3} \mathrm{H}(w - |\mathbf{r}|) \tag{7}$$

H is the Heaviside step function and the characteristic width w is the radius of the boundary. This distribution is not smooth and does thus not allow for the use of the derivative $\partial^\mu \phi$.

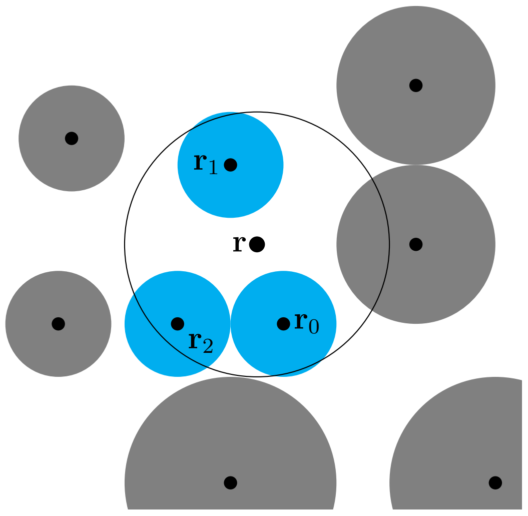
Another distribution, that is often more useful because of its smoothness, is the Gaussian distribution.

$$\phi(\mathbf{r}) = \frac{1}{\left(\sqrt{2\pi} w\right)^3} exp\left(-\frac{|\mathbf{r}|^2}{2w^2}\right) \tag{8}$$

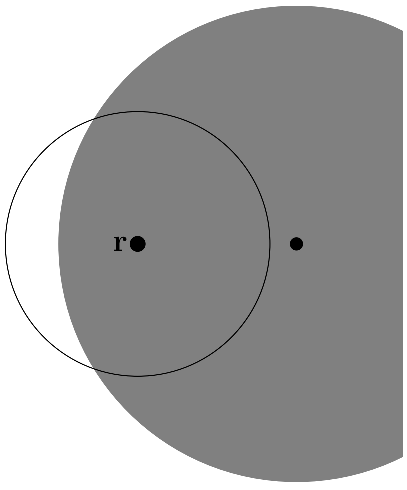
The characteristic width w is similar to the standard deviation. However, it is not precisely the same because this Gaussian is normalized in 3 dimensions.

The derivative of the Gaussian coarse graining function is:

$$\partial^\mu \phi(\mathbf{r}) = -\frac{r^\mu}{w^2} \phi(\mathbf{r}) \tag{9}$$



(a) Three particles contribute to the field at $\mathbf{r}$ if a Heaviside function is used for ϕ.



(b) A large particle almost completely covers the coarse graining region, yet it does not contribute to the the field at $\mathbf{r}$ at all.

Figure 1 – The coarse graining scale plays an important part in deciding whether coarse graining is a suitable technique for a given scenario.

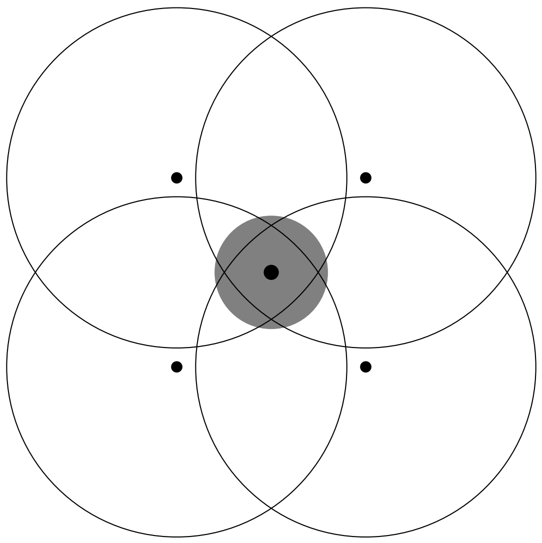
2.2.2 Implicitly Identical Particles

The characteristic coarse graining width w is a finite value, while the radii of the particles are completely ignored. However, the coarse graining functions used in this thesis are symmetric in the position of a given particle $\mathbf{r}_i$ and the position $\mathbf{r}$, at which the field is computed.

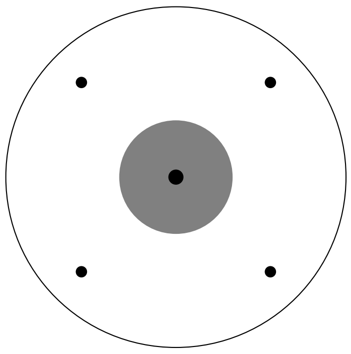
$$A\left(\mathbf{r}\right) = \sum_i A_i\, \phi\left(\left|\mathbf{r}_i - \mathbf{r}\right|\right) = \sum_i A_i\, \phi\left(\left|\mathbf{r} - \mathbf{r}_i\right|\right) \tag{10}$$

This means we can reinterpret the coarse graining function ϕ as the shape of the particles and in turn interpret the position $\mathbf{r}$ as a sampling point in the field that is the sum of all the individual particle fields. And, since the coarse graining function ϕ is fixed over the summation of the particles, this means that all particles are treated as having the exact same shape. Any information about particle size is discarded. This is illustrated in figure 2.

This might not be a problem if the range of particle sizes is small and the coarse graining scale w is chosen to appropriately represent the particle size. For systems with large variations in particle size however, this may well lead to undesirable features in the computed fields.



(a) Coarse graining only takes the particles center into account. It lies within the coarse graining region of 4 selected points.



(b) The particle will contribute its properties to the same 4 points. The size of the particle is misrepresented.

Figure 2 – A visual representation of the symmetry of ϕ in $\mathbf{r}$ and $\mathbf{r}_i$. The coarse graining process is equivalent to giving each particle the same shape and size. The circles may, for example, represent the boundary of a Heaviside function or the width of Gaussian.

2.2.3 Coarse Graining in a Regular Grid

Any continuous field contains an infinite number of points. For simulations it is thus imperative to reduce this number and use a description that requires only a finite number of data points. The process of converting the set of infinite data points in the continuum into a finite number of discrete values is commonly known as *discretization.*

The simplest method for doing this consists of definining a regular grid of points within a finite volume. Each point $\widetilde{\mathbf{r}}_j$ can then be uniquely identified by a set of integers λ^μ, representing its position along each axis. Given the grid spacings g^μ, the components of the point are then simply the product of the identifying integer and the spacing.

$$\widetilde{r}^\mu_j = \lambda^\mu g^\mu \tag{11}$$

The field at this point $\widetilde{A}_j$ is then computed by simply sampling the coarse grained field.

$$\widetilde{A}_j = A\left(\widetilde{\mathbf{r}}_j\right) \tag{12}$$

The set of samples is now finite and suitable for numerical experiments, but it is only defined in select points and is thus not a continuous field. To acquire a value at any other point and truly turn it into a field again, one has to make use of *interpolation.* This can provide a continuous field while only saving a finite amount of data. However, it is only an approximation of the underlying coarse grained field A.

The simplest interpolation scheme is called *nearest neighbour interpolation.* In it, the interpolated field $\widetilde{A}\left(\mathbf{r}\right)$ simply takes the value at the nearest grid point $\widetilde{\mathbf{r}}_j$. This creates identical cuboid regions C_j around each grid point, throughout which the value of the interpolated field is constant. These regions are called *cells* and their dimensions are precisely the same as the spacings g^μ between sampling points.

The *volume* V of such a cell is the product of its dimensions.

$$V = \prod_\mu g^\mu \tag{13}$$

And because the interpolated field is constant throughout the cell, the integral of the field within the cell transforms into a simple product.

$$\int_{C_j} d\mathbf{r}\, \widetilde{A}\left(\mathbf{r}\right) = \widetilde{A}_j V \tag{14}$$

If we compute the integral of the interpolated field over the entire space we get the *field total.* In this interpolation scheme it is simply the sum of the individual field values $\widetilde{A}_j$ times the cell volume, which is a constant.

$$\int_{\mathbf{R}^3} d\mathbf{r}\, \widetilde{A}\left(\mathbf{r}\right) = V \sum_j \widetilde{A}_j \tag{15}$$

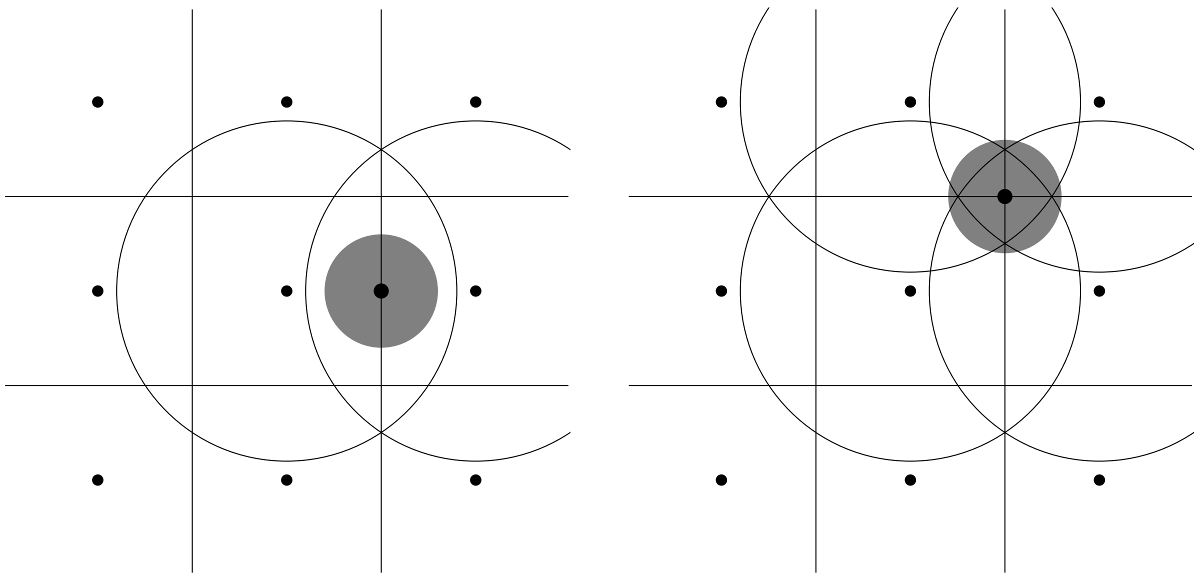
Here we make use of the assumption that the grid is infinite, so that the union of all cells equals the entire space.

$$\bigcup_j C_j = \mathbf{R}^3 \tag{16}$$

In practice the grid is finite, but it is a good approximation if the underlying field A is neglegible outside of the region enclosed by the grid.

Unfortunately, a problem arises, even in the case of an infinite grid. In equation 6 we demonstrated that the field total of the continuous coarse grained field $A\left(\mathbf{r}\right)$ matches the total value of the particles prior to coarse graining. This is, however, not generally the case for the approximation $\widetilde{A}\left(\mathbf{r}\right)$. That is, because the value of a cell $\widetilde{A}_j$ is treated as if it represented the mean of the underlying field $A\left(\mathbf{r}\right)$ throughout the cell, even though it does not. An example of how this can lead to practical problems is illustrated in figure 3.

If a greater degree of smoothness in the interpolated field is desired, other methods, e.g. trilinear or tricubic interpolation, may be used[7]. However, these might not guarantee conservation of the field totals either.


(a) The particle is sampled twice and thus contributes to the density of 2 cells.

(b) The particle is sampled 4 times and thus contributes to the density of 4 cells.

Figure 3 – A problem arises when a coarse graining function is calculated on the grid cells and the particles are approximated as point-like. Depending on a particle's position, its properties can be wrongly represented. If a Heaviside coarse graining function is used, moving the particle by just the distance of half a cell is enough to double the value it contributes to the field total. Note that we chose a typical size for the coarse graining function, but the effect persists for other sizes, as no space can be uniformly covered by spherically symmetric functions, with the exception of the sphere itself. A more homogeneous solution than the standard grid would be close packing of spheres.

2.3 Mean Value Discretization

Due to the problems with coarse graining mentioned in the previous sections, we propose an alternative method to calculate the fields. Instead of treating the particles as discrete elements we treat them as fields directly and omit the convolution with the coarse graining function.

A single particle can have any normalized distribution of its properties $\rho_i(\mathbf{r})$, much like $\phi(\mathbf{r})$, but defined per particle. This fixes the issue in section 2.2.2 of particles implicitly having the same shape in coarse graining. We limit ourselves to a particle having the same distribution ρ_i for all of its properties, though. While this is not a strict requirement for the theory, it does allow for more optimization of the algorithms later on. The continuous field $A(\mathbf{r})$ is then just the sum of all the particles' individual fields.

$$A(\mathbf{r}) = \sum_i A_i \rho_i(\mathbf{r} - \mathbf{r}_i) \tag{17}$$

If all distributions ρ_i are identical, this is equivalent to the continous coarse grained field. The normalization of the distributions ρ_i guarantees that the total properties of the particles are conserved when transitioning to the continuous field.

$$\int_{\mathbf{R}^3} d\mathbf{r} A(\mathbf{r}) = \sum_i A_i \int_{\mathbf{R}^3} d\mathbf{r}\, \rho_i(\mathbf{r}) = \sum_i A_i \tag{18}$$

We could now proceed to compute the values in a fixed grid, but the problem discussed in section 2.2.3 would still persist.

We propose the following to solve this issue. The value $\widetilde{A}_j$ in the grid point $\widetilde{\mathbf{r}}_j$ shall be defined as the mean value of the continuous field $A(\mathbf{r})$ throughout the cell C_j.

$$\widetilde{A}_j = \frac{1}{V}\int_{C_j} d\mathbf{r}\, A(\mathbf{r}) \tag{19}$$

We again use nearest neighbour interpolation to obtain a proper field $\widetilde{A}(\mathbf{r})$. Like before, we can now integrate over all of space by summing the contributions of the individual cells.

$$\int_{\mathbf{R}^3} d\mathbf{r}\, \widetilde{A}(\mathbf{r}) = \sum_j \int_{C_j} d\mathbf{r}\, \widetilde{A}(\mathbf{r}) = V \sum_j \widetilde{A}_j \tag{20}$$

The definition of the values $\widetilde{A}_j$ in equation 19 can now be plugged into this equation to relate the interpolated field total to the true field total.

$$\int_{\mathbf{R}^3} d\mathbf{r}\, \widetilde{A}(\mathbf{r}) = V \sum_j \left(\frac{1}{V}\int_{C_j} d\mathbf{r}\, A(\mathbf{r})\right) = \int_{\mathbf{R}^3} d\mathbf{r}\, A(\mathbf{r}) \tag{21}$$

The field total is hence conserved when transitioning from the true field $A(\mathbf{r})$ to the interpolated field $\widetilde{A}(\mathbf{r})$. Together with equation 18 this guarantees that the interpolated field total equals the total of the particles.

$$\int_{\mathbf{R}^3} d\mathbf{r}\, \widetilde{A}(\mathbf{r}) = \sum_i A_i \tag{22}$$

This fixes the issue with non-conservation of quantities in coarse graining, described in section 2.2.3, as intended.

2.3.1 Computing the Mean

The integral in equation 19 is impossible to compute in general, but becomes managable if we impose certain conditions on the true field $A(\mathbf{r})$. Specifically, we require the particles to be homogeneous, so that $\rho_i(\mathbf{r})$ can only take on a fixed value or 0. To maintain the normalization the fixed value is required to be $1/U_i$, where U_i is the particle's volume. This permits us to describe the particle shape as a region P_i instead.

When we now compute the value $\widetilde{A}_j$ in cell C_j using equations 17 and 19, the integral simplifies to the volume of intersection betweed the particle and the cell.

$$\widetilde{A}_j = \frac{1}{V}\int_{C_j} d\mathbf{r} \sum_i A_i \rho_i(\mathbf{r} - \mathbf{r}_i) = \sum_i \frac{A_i}{U_i V}\int_{C_j \cap P_i} d\mathbf{r} \tag{23}$$

Typically, we would like to model the particles as spheres. However, that would make the intersection between the cell and the particle that of a cuboid and a sphere. There is no analytical solution to this problem[8], so some approximations have to be made. Fortunately, the intersection of two arbitrary (but axis aligned) cuboids is easy to compute. We thus choose to approximate the particle as a set of (non-intersecting) cuboids Q_k.

$$P_i \longrightarrow \bigcup_k Q_k \tag{24}$$

We impose the additional condition that the volume of the approximation is equal to the volume of the original particle shape.

$$U_i = \int_{P_i} d\mathbf{r} = \int_{\bigcup_k Q_k} d\mathbf{r} = \sum_k \int_{Q_k} d\mathbf{r} \tag{25}$$

Since the particles are originally treated as spheres and their radius R_i is known, we can also make the volume explicit.

$$U_i = \frac{4}{3}\pi R_i^3 \tag{26}$$

With the cuboids approximating the particle, the intersection of cell and particle transforms into the following.

$$C_j \cap P_i \longrightarrow C_j \cap \left(\bigcup_k Q_k \right) = \bigcup_k \left(C_j \cap Q_k \right) \tag{27}$$

This now allows us to write equation 23 as a sum of integrals which are easy to evaluate.

$$\widetilde{A}_j = \sum_i \left(\frac{A_i}{U_i V} \int_{\bigcup_k (C_j \cap Q_k)} d\mathbf{r} \right) = \sum_i \left(\frac{A_i}{U_i V} \sum_k \int_{C_j \cap Q_k} d\mathbf{r} \right) \tag{28}$$

The integrals are the volume of intersection between two cuboids. We introduce the vectors $\mathbf{C}_-$ and $\mathbf{C}_+$, as well as $\mathbf{Q}_-$ and $\mathbf{Q}_+$ as the minimum and maximum bounds of the cuboids C and Q along the grid axes. Given that the cuboids intersect, the integral of intersection can be written as:

$$\int_{C \cap Q} d\mathbf{r} = \prod_\mu \left(\min\left(C_+^\mu, Q_+^\mu\right) - \max\left(C_-^\mu, Q_-^\mu\right) \right) \tag{29}$$

An intersection occurs if and only if each of the factors in this product is positive. We can conveniently modify the equation by adding a Heaviside function to take this into account and evaluate to 0 if there is no intersection.

$$\int_{C \cap Q} d\mathbf{r} = \prod_\mu \left(\min\left(C_+^\mu, Q_+^\mu\right) - \max\left(C_-^\mu, Q_-^\mu\right) \right) \mathrm{H}\left(\min\left(C_+^\mu, Q_+^\mu\right) - \max\left(C_-^\mu, Q_-^\mu\right) \right) \tag{30}$$

While lengthy to write, it is very easy to actually evaluate. For the sake of readability we will continue to write the integral, though.

With an arbitrary number of cubes, any particle shape P_i can be approximated arbitrarily closely. However, we limit ourselves to using only a single cube for every particle, as the speed of the computation is more relevant for our purposes than the accuracy. With this single cube approximation $P_i = Q_i$, the final expression is rather compact.

$$\widetilde{A}_j = \frac{1}{V} \sum_i \frac{A_i}{U_i} \int_{C_j \cap Q_i} d\mathbf{r} \tag{31}$$

2.4 Discrete Element Method

In this thesis we will apply principles of the discrete element method to granular media. More specifically, the subject of study are media of perfectly spherically symmetric particles. Contacts between particles are not pointlike, but particles are permitted a finite overlap. This overlap, however, will play into the dynamics of the system. The following subsections will provide information on the particle and contact properties of interest. It should be noted that, within the context of this report, everything that is calculated by the external physics engine, is considered as fundamental. This does not reflect what is considered fundamental in the wider physics community.

2.4.1 Relevant Fundamental Properties

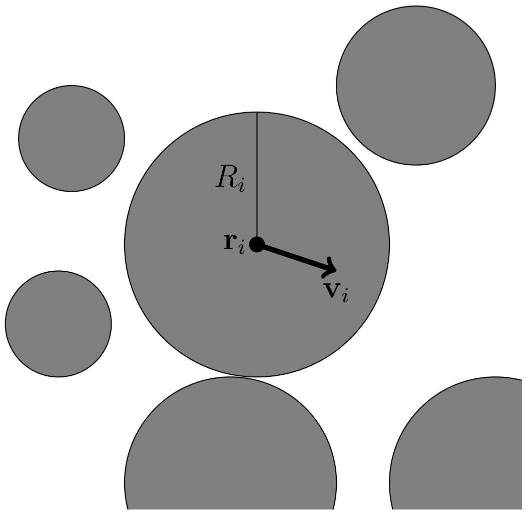
The number of fundamental particle properties required to calculate a vast number of physical fields is very small. Since there is usually multiple particles we use an index to identify the particle. For this thesis the fundamental particle properties of interest are:

Particle Position	$\mathbf{r}_i$
Particle Velocity	$\mathbf{v}_i$
Particle Radius	R_i
Particle Mass	m_i

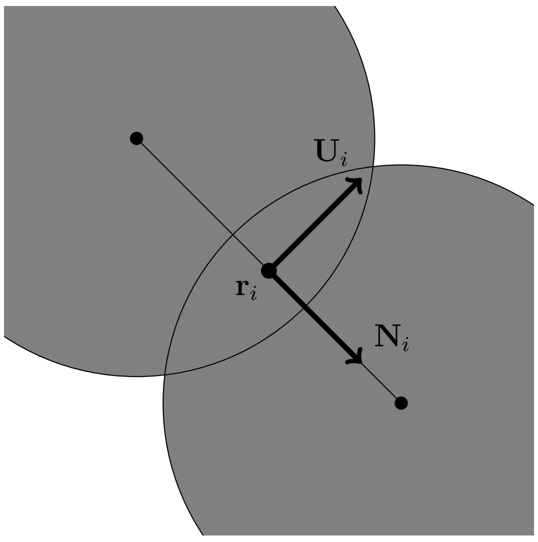
Additionally, the contacts between particles are also treated as fundamental. The fundamental contact properties of interest are:

Contact Position	$\mathbf{r}_i$
Contact Normal	$\mathbf{N}_i$
Contact Tangent U	$\mathbf{U}_i$
Contact Tangent V	$\mathbf{V}_i$
Contact Local Force	$\mathbf{f}_i$

The quantities for which there is an intuitive spatial understanding are shown in figure 4. Other quantities will be defined in the following sections. Also, note that that for each contact i, the involved particle indices are also accessible, such that we can resolve $\mathbf{r}_i = \mathbf{r}_{ab}$ to clarify that the contact i is between particles a and b.



(a) The fundamental properties of a particle. The mass m_i is an intrinsic scalar value that can not be drawn.



(b) The fundamental properties of a contact. The tangent $\mathbf{V}_i$ is perpendicular to both the normal $\mathbf{N}_i$ and tangent $\mathbf{U}_i$ and points out of the plane.

Figure 4 – The particles and contacts have several properties that correspond to spatial dimensions and can thus be visualized.

2.4.2 Derived Quantities

In addition to the fundamental particle and contact properties, there are several derived quantities that are of interest[1, 2].

The particle *displacement* is the difference between a particle's current position and its position at some reference time τ.

$$\mathbf{u}_i\left(t\right) = \mathbf{r}_i\left(t\right) - \mathbf{r}_i\left(\tau\right) \tag{32}$$

The particle *momentum* is the product of the particle's velocity and its mass.

$$\mathbf{p}_i = m_i \mathbf{v}_i \tag{33}$$

A particle's *kinetic stress* is a helpful property to calculate the kinetic stress field later on. It is defined as the negative of the outer product of a particle's momentum and velocity.

$$\left(\sigma^{\mathrm{k}}\right)_i^{\mu\nu} = -p_i^\mu v_i^\nu = -m_i v_i^\mu v_i^\nu \tag{34}$$

Instead of writing this definition in terms of components, it can be expressed idiomatically. This helps with readability as the number of indices is reduced.

$$\left(\boldsymbol{\sigma}^{\mathrm{k}}\right)_i = -\left(\mathbf{p}_i \otimes \mathbf{v}_i\right) = -m_i\left(\mathbf{v}_i \otimes \mathbf{v}_i\right) \tag{35}$$

A particles's *mass discplacement* is defined as the product of the particle's displacement and its mass. Like the particle's kinetic stress this is only used as a helpful intermediate quantity.

$$\left(\mathbf{u}^{\mathrm{m}}\right)_i = m_i \mathbf{u}_i \tag{36}$$

A contact's *branch vector* is the difference of the positions of the involved particles. Resolving the contact index i to the pair of indices of involved particles (a, b), as described in section 2.4.1, this can be written as:

$$\mathbf{B}_i = \mathbf{B}_{ab} = \mathbf{r}_b - \mathbf{r}_a \tag{37}$$

A contact's *local force* is the contact force's components in a reference frame consisting of the contact normal $\mathbf{N}_i$ and tangents $\mathbf{U}_i$ and $\mathbf{V}_i$. To calculate the *contact force* in the global frame of reference we multiply each component by the corresponding vector and add the results.

$$\mathbf{F}_i = f_i^0 \mathbf{N}_i + f_i^1 \mathbf{U}_i + f_i^2 \mathbf{V}_i \tag{38}$$

The *contact stress*, like a particle's kinetic stress and mass displacement, is a helpful intermediate quantity. It is used in the calculation of the contact stress field later on. It is defined as the negative of the outer product of the contact force and the branch vector.

$$\left(\sigma^{\mathrm{c}}\right)_i^{\mu\nu} = -F_i^\mu B_i^\nu \tag{39}$$

And just like the kinetic stress, this contact stress can also be written idiomatically to reduce the number of indices.

$$\left(\boldsymbol{\sigma}^{\mathrm{c}}\right)_i = -\left(\mathbf{F}_i \otimes \mathbf{B}_i\right) \tag{40}$$

2.5 The Fields

2.5.1 Coarse Grained Fields

The particle properties in the previous section were defined deliberately such that the expressions for the following fields all take the form of equation 1. This has the practical consequence that it keeps the number of necessary algorithms, which we have to develop later on, to a minimum.

$$m(\mathbf{r}) = \sum_i \phi(\mathbf{r}_i - \mathbf{r})\; m_i \tag{41}$$

$$\mathbf{p}(\mathbf{r}) = \sum_i \phi(\mathbf{r}_i - \mathbf{r})\; \mathbf{p}_i \tag{42}$$

$$\left(\boldsymbol{\sigma}^{\mathrm{k}}\right)(\mathbf{r}) = \sum_i \phi(\mathbf{r}_i - \mathbf{r})\; \left(\boldsymbol{\sigma}^{\mathrm{k}}\right)_i \tag{43}$$

$$\left(\mathbf{u}^{\mathrm{m}}\right)(\mathbf{r}) = \sum_i \phi(\mathbf{r}_i - \mathbf{r})\; \left(\mathbf{u}^{\mathrm{m}}\right)_i \tag{44}$$

These are the mass, momentum, kinetic stress and mass displacement fields. Because they are continuous, they correspond to densities, which is henceforth implied when fields are mentioned. This is done to avoid confusion, as density in physics often refers to the mass density alone.

Apart from the fields that can be written in this simple form, there are some more complicated fields. They either depend on some of these simple fields, or they require the gradient of the coarse graining function ϕ.

Even though the particles have a velocity $\mathbf{v}_i$, the velocity field is not defined by putting these velocities into equation 1. Instead, it is defined as the quotient of the momentum and mass fields.

$$\mathbf{v}(\mathbf{r}) = \frac{\mathbf{p}(\mathbf{r})}{m(\mathbf{r})} \tag{45}$$

This velocity field can now be used in combination with the particle velocities to define the *granular temperature.*

$$T(\mathbf{r}) = \sum_i \phi(\mathbf{r}_i - \mathbf{r})\; \left(\mathbf{v}_i - \mathbf{v}(\mathbf{r})\right)^2 \tag{46}$$

Similarly to the velocity field, the displacement field is defined as the quotient of the mass displacement field and the mass field.

$$\mathbf{u}(\mathbf{r}) = \frac{\left(\mathbf{u}^{\mathrm{m}}\right)(\mathbf{r})}{m(\mathbf{r})} \tag{47}$$

The gradient of the displacement field and the gradient of the velocity field can be written in terms of the coarse graining function as well. We call these the *deformation* tensor and the *deformation rate* tensor.

$$D^{\mu\nu}(\mathbf{r}) = \frac{\partial u^\mu}{\partial r^\nu}(\mathbf{r}) = \frac{\sum_{ij} m_i m_j \left(u_i^\mu - u_j^\mu\right)\left(\partial^\nu \phi(\mathbf{r}_i - \mathbf{r})\right)\phi(\mathbf{r}_j - \mathbf{r})}{m(\mathbf{r})^2} \tag{48}$$

$$D'^{\mu\nu}(\mathbf{r}) = \frac{\partial v^\mu}{\partial r^\nu}(\mathbf{r}) = \frac{\sum_{ij} m_i m_j \left(v_i^\mu - v_j^\mu\right)\left(\partial^\nu \phi(\mathbf{r}_i - \mathbf{r})\right)\phi(\mathbf{r}_j - \mathbf{r})}{m(\mathbf{r})^2} \tag{49}$$

In the approximation of small variations in the mass field $\nabla m \approx \mathbf{0}$ these expressions simplify. The numerators of these expressions can be calculated without needing any other fields, but they depend on the derivative of the coarse graining function $\partial^\nu \phi$.

$$D^{\mu\nu}(\mathbf{r}) \approx \frac{\sum_i m_i u_i^\mu \left(\partial^\nu \phi\left(\mathbf{r}_i - \mathbf{r}\right)\right)}{m(\mathbf{r})} \tag{50}$$

$$D'^{\mu\nu}(\mathbf{r}) \approx \frac{\sum_i m_i v_i^\mu \left(\partial^\nu \phi\left(\mathbf{r}_i - \mathbf{r}\right)\right)}{m(\mathbf{r})} \tag{51}$$

We again provide idiomatic expressions for the tensors to simplify calculations later on.

$$\mathbf{D}(\mathbf{r}) \approx \frac{\sum_i \left(\mathbf{u}^{\mathrm{m}}\right)_i \otimes \left(\nabla\phi\left(\mathbf{r}_i - \mathbf{r}\right)\right)}{m(\mathbf{r})} \tag{52}$$

$$\mathbf{D}'(\mathbf{r}) \approx \frac{\sum_i \mathbf{p}_i \otimes \left(\nabla\phi\left(\mathbf{r}_i - \mathbf{r}\right)\right)}{m(\mathbf{r})} \tag{53}$$

The *strain* and *strain rate* tensors are defined by the symmetrization of the deformation and deformation rate respectively. In practice we will use the approximations for a small mass gradient.

$$\varepsilon^{\mu\nu}(\mathbf{r}) = \frac{1}{2}\left(D^{\mu\nu}(\mathbf{r}) + D^{\nu\mu}(\mathbf{r})\right) \tag{54}$$

$$\varepsilon'^{\mu\nu}(\mathbf{r}) = \frac{1}{2}\left(D'^{\mu\nu}(\mathbf{r}) + D'^{\nu\mu}(\mathbf{r})\right) \tag{55}$$

Additionally to all these fields based purely on single particle properties, we define the contact stress tensor. Using the previously defined stress of a contact $\left(\boldsymbol{\sigma}^{\mathrm{c}}\right)_i$ and given a straight line Γ_i between the centers of the particles involved in the contact i, it is defined as follows.

$$\left(\boldsymbol{\sigma}^{\mathrm{c}}\right)(\mathbf{r}) = \sum_i \left(\left(\boldsymbol{\sigma}^{\mathrm{c}}\right)_i \int_{\Gamma_i} \phi\left(\mathbf{s} - \mathbf{r}\right) d\mathbf{s}\right) \tag{56}$$

This expression can be simplified if the contacts, just like the particles, are approximated as point-like. We can do this under the assumption that the length of the branch vector is much smaller than the characteristic coarse graining width $|\mathbf{B}_i| \ll w$.

$$\left(\boldsymbol{\sigma}^{\mathrm{c}}\right)(\mathbf{r}) = \sum_i \phi\left(\mathbf{r}_i - \mathbf{r}\right) \left(\boldsymbol{\sigma}^{\mathrm{c}}\right)_i \tag{57}$$

This gives the expression for the contact stress field the same form as any of the simple particle fields, but it uses the contact position $\mathbf{r}_i$ and contact stress $\left(\boldsymbol{\sigma}^{\mathrm{c}}\right)_i$.

With the help of the kinetic stress and the contact stress we can finally define the total *stress*, which is simply the sum of both.

$$\boldsymbol{\sigma}(\mathbf{r}) = \left(\boldsymbol{\sigma}^{\mathrm{k}}\right)(\mathbf{r}) + \left(\boldsymbol{\sigma}^{\mathrm{c}}\right)(\mathbf{r}) \tag{58}$$

Using this stress tensor, we can define some more properties which can be helpful in the physical analysis of systems. They are the *pressure* P, the *stress deviator* $\overline{\boldsymbol{\sigma}}$ and the *von Mises stress* $\left(\sigma^{\mathrm{vm}}\right)$.

$$P(\mathbf{r}) = -\frac{1}{3}\mathrm{Tr}\left(\boldsymbol{\sigma}(\mathbf{r})\right) \tag{59}$$

$$\overline{\boldsymbol{\sigma}}(\mathbf{r}) = \boldsymbol{\sigma}(\mathbf{r}) + P(\mathbf{r})\, \mathbf{I} \tag{60}$$

$$\left(\sigma^{\mathrm{vm}}\right)(\mathbf{r}) = \sqrt{\frac{3}{2}\sum_{\mu\nu} \overline{\sigma}^{\mu\nu}(\mathbf{r})\, \overline{\sigma}^{\mu\nu}(\mathbf{r})} \tag{61}$$

$\mathbf{I}$ denotes the identity matrix.

2.5.2 Fields with Mean Value Discretization

To define the simple fields with the alternative approach, we simply use equation 31, instead of equation 1.

$$m\left(\tilde{\mathbf{r}}_j\right) = \frac{1}{V} \sum_i \frac{m_i}{U_i} \int_{C_j \cap Q_i} d\mathbf{r} \tag{62}$$

$$p\left(\tilde{\mathbf{r}}_j\right) = \frac{1}{V} \sum_i \frac{p_i}{U_i} \int_{C_j \cap Q_i} d\mathbf{r} \tag{63}$$

$$\left(\boldsymbol{\sigma}^k\right)\left(\tilde{\mathbf{r}}_j\right) = \frac{1}{V} \sum_i \frac{\left(\boldsymbol{\sigma}^k\right)_i}{U_i} \int_{C_j \cap Q_i} d\mathbf{r} \tag{64}$$

$$\left(\boldsymbol{u}^m\right)\left(\tilde{\mathbf{r}}_j\right) = \frac{1}{V} \sum_i \frac{\left(\boldsymbol{u}^m\right)_i}{U_i} \int_{C_j \cap Q_i} d\mathbf{r} \tag{65}$$

In equation 57 of the coarse graining process, the contact stress is treated as point-like and thus obeys the same form of an equation as the particle properties. Using this concept as a guideline, we also treat the contact similarly to a particle, in that we define the contact stress with the same equation as the fields derived from the particle properties.

$$\left(\boldsymbol{\sigma}^c\right)\left(\tilde{\mathbf{r}}_j\right) = \frac{1}{V} \sum_i \frac{\left(\boldsymbol{\sigma}^c\right)_i}{U_i} \int_{C_j \cap Q_i} d\mathbf{r} \tag{66}$$

The cuboid Q_i and volume U_i are however not defined for contacts. A possible choice is to treat the contact as a cube with the contact depth as its side length. Alternatively, one can simply use a small constant side length, compared to the grid spacings g^μ, to treat the contacts as point like.

The definition of the velocity, displacement and the fields derived from the stresses remain unchanged.

$$\mathbf{v}\left(\tilde{\mathbf{r}}_j\right) = \frac{\mathbf{p}\left(\tilde{\mathbf{r}}_j\right)}{m\left(\tilde{\mathbf{r}}_j\right)} \tag{67}$$

$$\mathbf{u}\left(\tilde{\mathbf{r}}_j\right) = \frac{\left(\mathbf{u}^{\mathrm{m}}\right)\left(\tilde{\mathbf{r}}_j\right)}{m\left(\tilde{\mathbf{r}}_j\right)} \tag{68}$$

$$\boldsymbol{\sigma}\left(\tilde{\mathbf{r}}_j\right) = \left(\boldsymbol{\sigma}^k\right)\left(\tilde{\mathbf{r}}_j\right) + \left(\boldsymbol{\sigma}^c\right)\left(\tilde{\mathbf{r}}_j\right) \tag{69}$$

$$P\left(\tilde{\mathbf{r}}_j\right) = -\frac{1}{3}\mathrm{Tr}\left(\boldsymbol{\sigma}\left(\tilde{\mathbf{r}}_j\right)\right) \tag{70}$$

$$\overline{\boldsymbol{\sigma}}\left(\tilde{\mathbf{r}}_j\right) = \boldsymbol{\sigma}\left(\tilde{\mathbf{r}}_j\right) + P\left(\tilde{\mathbf{r}}_j\right)\mathbf{I} \tag{71}$$

$$\left(\boldsymbol{\sigma}^{\mathrm{vm}}\right)\left(\tilde{\mathbf{r}}_j\right) = \sqrt{\frac{2}{3}\sum_{\mu\nu} \overline{\sigma}^{\mu\nu}\left(\tilde{\mathbf{r}}_j\right)\overline{\sigma}^{\mu\nu}\left(\tilde{\mathbf{r}}_j\right)} \tag{72}$$

The definition of the deformation tensor and the deformation rate tensor change slightly. In the coarse graining process they rely on the gradient $\nabla\phi$ of a Gaussian coarse graining function ϕ. That is not an option in this alternative approach, so the symmetric finite difference gradient is used instead.

$$D^{\mu\nu}\left(\tilde{\mathbf{r}}_j\right) = \frac{\Delta u^\mu}{\Delta r^\nu}\left(\tilde{\mathbf{r}}_j\right) \tag{73}$$

$$D'^{\mu\nu}\left(\tilde{\mathbf{r}}_j\right) = \frac{\Delta v^\mu}{\Delta r^\nu}\left(\tilde{\mathbf{r}}_j\right) \tag{74}$$

Using these new definitions, the strain tensor and strain rate tensor remain otherwise unchanged.

$$\varepsilon^{\mu\nu}\left(\widetilde{\mathbf{r}}_j\right) = \frac{1}{2}\left(D^{\mu\nu}\left(\widetilde{\mathbf{r}}_j\right) + D^{\nu\mu}\left(\widetilde{\mathbf{r}}_j\right)\right) \tag{75}$$

$$\varepsilon'^{\mu\nu}\left(\widetilde{\mathbf{r}}_j\right) = \frac{1}{2}\left(D'^{\mu\nu}\left(\widetilde{\mathbf{r}}_j\right) + D'^{\nu\mu}\left(\widetilde{\mathbf{r}}_j\right)\right) \tag{76}$$

2.6 Machine Learning

Machine learning is an extensive field of active research. Especially artificial neural networks, which originally aimed to mimic the biological processes of the brain[9], are a fairly recent development. Even though some aspects of these neural networks still lack mathematical rigour[10], in practice they are already widely used for many purposes.

In the following sections we will focus purely on the mathematical model, ignoring the biologic aspect altogether.

2.6.1 Artifical Neuron

An artifical neuron is the smallest component of artifical neural networks. It is a function that maps a number of input values to a single output value[11]. A graphical representation of an artificial neuron is shown in figure 5. It is calculated in two steps:

First, the weighted sum Σ of the input variables is computed. The weights w_i are what changes during the training process, which is explained in later sections.

In the second step, this sum is used as the input to some activation function Φ. The complete expression for the output of the neuron is then:

$$y = \Phi\left(\sum_{i=0}^{N} w_i x_i\right) \tag{77}$$

The 0th input is usually fixed to be $x_0 = 1$. The weight of this input w_0 is then also called the *bias* of the neuron. It is independent of the variable inputs.

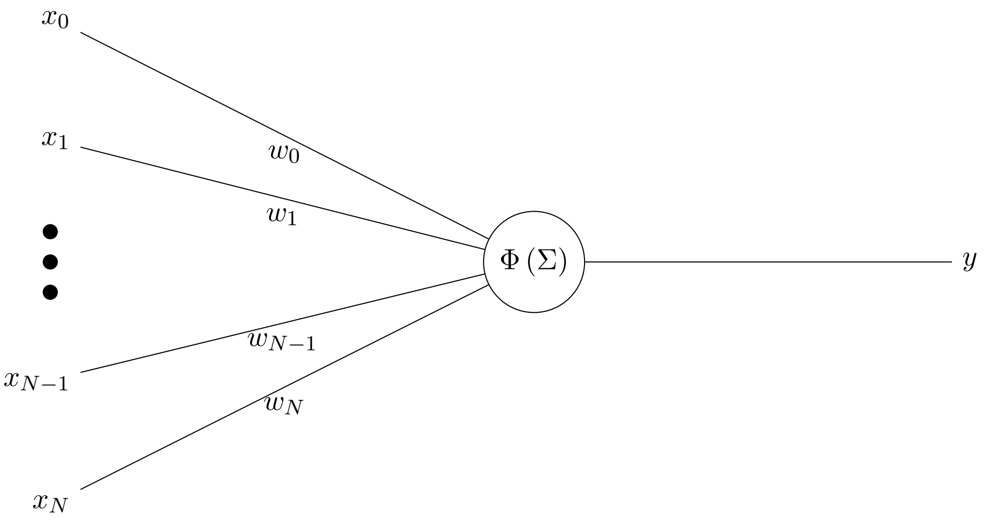
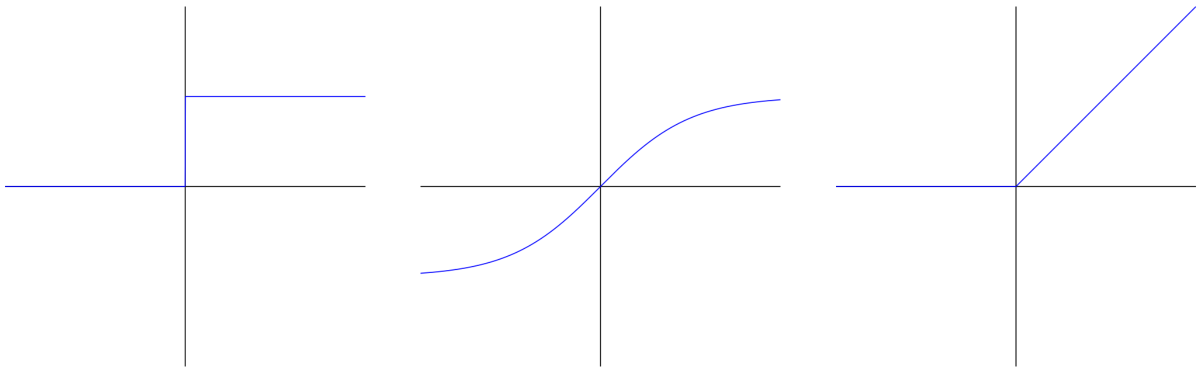


Figure 5 – An artifical neuron that takes $N+1$ input variables x_i and produces a single output value y. The weights w_i are intrinsic properties of the neuron and subject to change during the training process.

2.6.2 Activation Function

The activation function Φ can, in theory, be any function. However, certain functions have proven more useful than others in the context of machine learning.



(a) Heaviside Function **(b)** Hyperbolic Tangent **(c)** Rectified Linear Unit ($ReLU$)

Figure 6 – Some of the most popular activation functions. They need not necessarily be continuous or bounded, although these properties can simplify training.

The simplest choice, aside from a constant, is the identity $\Phi(\Sigma) = \Sigma$. A neuron with such an activation function is then simply a linear combination of inputs.

$$y = \sum_{i=0}^{N} w_i x_i \tag{78}$$

While not entirely without use, its practicality is fairly limited. Even in networks with many layers, this activation function will, at best, be able to perform linear regression. That is, because a linear combination of linear combinations leads to a simple linear combination overall.

$$\begin{aligned} y &= w_0 (u_0 x_0 + u_1 x_1) + w_1 (v_0 x_0 + v_1 x_1) \\ &= (w_0 u_0 + w_1 v_0) x_0 + (w_0 u_1 + w_1 v_1) x_1 \end{aligned} \tag{79}$$

For this reason, the identity is rarely used on its own as an activation function in neural networks.

However, with just a small tweak we can generate the most widely used[12] activation function of today. If the input is positive $\Sigma > 0$ it simply returns the identity. However, if that is not the case it returns 0.

$$\Phi(\Sigma) = \begin{cases} \Sigma & \text{if } \Sigma > 0 \\ 0 & \text{otherwise} \end{cases} \tag{80}$$

This activation function is called the *rectified linear unit*($ReLU$). It is plotted alongside some other popular activation functions in figure 6.

Artifical neurons with this $ReLU$ activation function are the basis for the data driven model in the final section of this thesis.

2.6.3 Neural Networks

Networks of multiple artificial neurons are called *artificial neural networks*, or simply *neural networks*. They are of particular interest because they have the capacity to *learn* relations between input and output data without requiring an explicit definition of a regression function[13]. However, in many cases they are still closely related to statistical models[14].

In this thesis we will deal exclusively with fully connected feed forward networks, such as the one shown in figure 7. This type of network is also called *multilayer perceptron.* (Although this term is sometimes reserved for models with a Heaviside activation function.)

Fully connected means that each node in a given layer serves as an input to every node in the next layer; There is a connection between each pair of nodes in two adjacent layers.

Feed forward means that there is a distinct order to the layers; No nodes in a given layer contribute as input to any node in any previous layer.

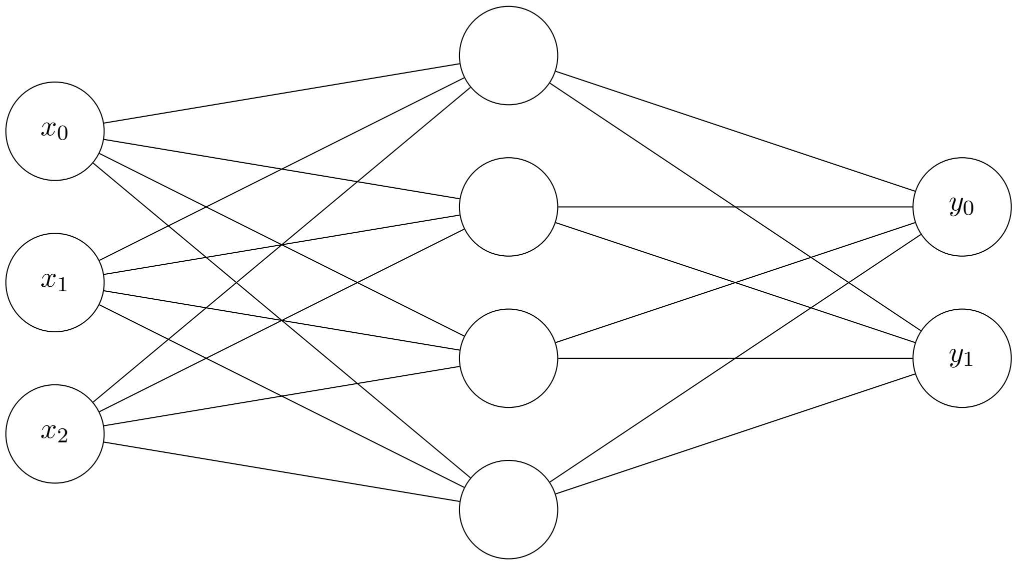


Figure 7 – A simple feed forward network that propagates the input values $\mathbf{x}$ from the left to the output values $\mathbf{y}$ on the right. A bias is not drawn explicitly, but is usually implied to exist.

In such a neural network the nodes are organized into distinct *layers.* Each layer has a *width*, which is the number of neurons in that layer. The weights in a network are also collectively called the *parameters*, as they change during the training process. The number of those parameters is commonly used as a simple metric for the complexity of a given network and can range in the hundreds of billions for complex tasks[15]. However, in cases of large networks like these, it often becomes difficult to explain how the network makes a decision[16]. Fortunately, a much smaller network will suffice for the data driven model in section 5.

2.6.4 Training a Network

To train a given network, a *loss function* $\mathcal{L}$ is required. This loss depends on the networks parameters and training is a process that adjusts these parameters to minimize the loss function.

The loss function can sometimes be formulated solely in terms on the network's prediction $\mathbf{y}^P$. In that case the training paradigm is called *unsupervised training.* The loss function in unsupervised learning strongly depends on the exact problem that the network is applied to.

In other cases the training dataset might consist of pairs of input features $\mathbf{x}$ and the true output labels $\mathbf{y}^\mathrm{T}$. This then allows for the definition of a loss function in terms of both the network's prediction $\mathbf{y}^\mathrm{P}$ and the true labels $\mathbf{y}^\mathrm{T}$. This training paradigm is then called *supervised training.* In supervised learning there is a small number of loss functions that can be applied to solve a wide variety of problems. A typical example for such a loss function is the mean square error between the network's prediction $\mathbf{y}^{\mathrm{P}}$ and the true labels $\mathbf{y}^{\mathrm{T}}$.

$$\mathcal{L}_i = \frac{1}{N}\sum_{j=1}^{N}\left(y_j^{\mathrm{P}} - y_j^{\mathrm{T}}\right)^2 \tag{81}$$

This mean square error is useful for most regression problems and will be used in our model later on. It is defined per *sample*, i.e. per pair of input and output data.

During the training process, the only variables are the network parameters $\mathbf{w}$. They occur in the loss function as part of $\mathbf{y}^{\mathrm{P}}$, as it is simply the propagation of the inputs $\mathbf{x}$ through the network with weights $\mathbf{w}$ (see equation 77). The weights are changed such as to minimize the average loss over the dataset.

$$\mathcal{L}\left(\mathbf{w}\right) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_i\left(\mathbf{w}\right) \tag{82}$$

2.6.5 Gradient Descent

There exists several methods to change the weights of a network to achieve minimum loss. A very effective method is to compute the gradient of the loss with respect to the network parameters and change them accordingly. Such methods are collectively referred to as *gradient descent*, but there are many variants of it.

Batch gradient descent is an iterative algorithm, that uses the entire dataset in each iteration. A single iteration of it is defined as[17]:

$$\mathbf{w}' = \mathbf{w} - \alpha\,\nabla\mathcal{L}\left(\mathbf{w}\right) = \mathbf{w} - \frac{\alpha}{N}\sum_{i=1}^{N}\nabla\mathcal{L}_i\left(\mathbf{w}\right) \tag{83}$$

$\mathbf{w}$ are the current model parameters, whereas $\mathbf{w}'$ are the parameters after the iteration. $\nabla\mathcal{L}$ is the gradient of loss with respect to the model parameters $\mathbf{w}$. α is simply a positive number, often called the *learning rate*. If this learning rate is chosen appropriately, convergence of the method to a local minimum is guaranteed[18].

Because it can be expensive for large datasets to compute the gradient of every $\mathcal{L}_i$ individually, a stochastic counterpart has been developed. Stochastic gradient descent makes an iteration for each sample as opposed to an iteration for the entire dataset.

$$\mathbf{w}' = \mathbf{w} - \alpha\,\nabla\mathcal{L}_i \tag{84}$$

Once the entire dataset has been exhausted, it can be shuffled and the next iteration can begin. Such a pass over the entire training dataset is called an *epoch*. It can be shown with few extra assumptions, that this method also converges (almost surely) to a local minimum[19].

In practice one can often find a hybrid approach that creates *minibatches*, consisting of a small number of samples, compared to the size of the entire dataset. This results in fast computation times, because it can utilize modern computer hardware more effectively.

These basic gradient descent methods have since been superseded by more advanced variants that often converge faster or require less manual tweaking. One such advanced optimizer, called *Adam*[20], is used in the data driven model in section 5. The details of how exactly it works go beyond the scope of this thesis.

3 Implementation

The first goal of this thesis was to implement coarse graining in a more efficient way than the previos implementations of the work group. To accomplish this, a combination of changes in platform and algorithm was devised.

The code is open source and available at [https://gitlab.com/agentklaas/agxfern.git] and its dependencies.

3.1 Platform Choices

3.1.1 Physics Engine

AGX Dynamics was chosen as the physics engine. This was a choice made for convenience as the work group is familiar with this engine and applications of the previous coarse graining implementation made use of this engine as well. However, all low level functions (i.e. the previously discussed algorithms) operate on raw data arrays. These can easily be created for any other physics engine as well.

It is only some convenience functions, like automatically importing and formatting the particle data, that are exclusively available for *AGX Dynamics*.

3.1.2 Programming Languages

The final implementation was mainly going to be used in *Python*, but to achieve the best possible performance, the low level routines had to be written in a compiled language. *C++* was chosen for that role as it is widely supported and offers great performance.

Python bindings were then created on top of this implementation, using the *pybind11* library. It was chosen because it is a lightweight, header-only, and easy to use binding library. The resulting package is a single file, which also helps with portability.

Lastly, while development was done on *Linux*, both *C++* and *Python* are supported by all major operating systems and no dependencies specific to a single operating system were used.

3.1.3 Machine Learning Library

TensorFlow is used as the platform for machine learning in the last parts of this thesis. It is responsible for the model, the training and validation process, as well as the live displacement prediction using the trained model. Built on top of *TensorFlow*, the *Keras* API provides an easy to use interface.

3.2 Algorithms

Some intermediate quantities in section 2 may have appeared overly convoluted. However, in actually computing the results we can now reap the benefits of defining them in the precise way that we did. Specifically, we reduce the number of necessary algorithms to just a handful.

In the following sections we will to present the algorithms in a clear and readable format and analyze their computational complexity. In practice, additional algebraic and programmatic optimizations are made. These do not affect the computational complexity.

3.2.1 Coarse Graining

Let's recall the fundamental coarse graining equation.

$$A\left(\mathbf{r}\right) = \sum_i A_i \phi\left(\mathbf{r}_i - \mathbf{r}\right) \tag{85}$$

The first algorithm will handle the calculation of this equation for all the points in a given grid with spacings $\mathbf{g}$. This algorithm is then used to calculate many of the fields defined in section 2.5.1.

In a naive implementation, one would have to loop through all pairs of particles and cells. This is slow if both the number of particles and the number of cells are large. Knowing that the coarse graining function ϕ falls off at a distance, we can introduce a cutoff region, beyond which the values are neglegibly small. We then only need to compute the value of ϕ at the cells within the cutoff region. For a Heaviside distribution the cutoff is simply the radius $c = w$. For a Gaussian distribution the cutoff was chosen as $c = 3w$. To make the notation of the algorithms simpler we also introduce the vectorized cutoff $\mathbf{c} = (c, c, c)$.

The loops in the algorithm can theoretically be put into any order, but choosing (particle → cell → property) from outermost to innermost loop permits the most reusing of temporary quantities. Especially ϕ needs to be computed only once per particle-cell pair, and can then be used for all the properties.

```
for ($i = 0$ to $i = N - 1$)
{
    $\boldsymbol{\lambda}_- = $ int$((\mathbf{r}_i - \mathbf{c})/\mathbf{g})$
    $\boldsymbol{\lambda}_+ = $ int$((\mathbf{r}_i + \mathbf{c})/\mathbf{g})$
    for ($\boldsymbol{\lambda} = \boldsymbol{\lambda}_-$ to $\boldsymbol{\lambda} = \boldsymbol{\lambda}_+$)
    {
        $\Phi = \phi(\widetilde{\mathbf{r}}_{\boldsymbol{\lambda}} - \mathbf{r}_i)$
        for ($k = 0$ to $k = M - 1$)
        {
            $\widetilde{A}^k_{\boldsymbol{\lambda}} \mathrel{+}= A^k_i \Phi$
        }
    }
}
```

Algorithm 1 – The coarse graining algorithm. N is the number of particles. $\mathbf{c}$ is the vectorized cutoff. $\mathbf{g}$ is the grid spacing vector. The `int` keyword represents rounding down to the nearest integer. $\boldsymbol{\lambda}_-$ and $\boldsymbol{\lambda}_+$ denote the minimum and maximum cell within the cutoff region. The index k is used to denote any of the properties A^k_i of particle i and the corresponding field $\widetilde{A}^k_{\boldsymbol{\lambda}}$ at the grid point $\boldsymbol{\lambda}$. M is the number of such properties. $\mathbf{r}_i$ is the particle's position and $\widetilde{\mathbf{r}}_{\boldsymbol{\lambda}}$ is the grid position.

By checking the bounds of the loops, the computational complexity is easily found. Let N be the number of discrete elements (either particles or contacts) and M be the number of properties. The complexity is then:

$$\mathcal{O}(NM) \tag{86}$$

The underlying concept of coarse graining only makes sense, when the coarse graining width is of a similar size as the grid spacings. If it is too large, the fields are unnecessarily smoothed and information is lost. If it is too small, then the true field is sampled too inhomogeneously and deviations of the field totals become large. Thus, the coarse graining width w should always be chosen to be of a similar size as the cell size. So if the resolution of the grid increases and the cell size decreases, then so does the coarse graining width and correspondingly the cutoff region. Hence, the number of cells within the cutoff region is approximatly constant and the total number of cells does not appear in the computational complexity.

3.2.2 Coarse Graining Derivatives

The next algorithm calculates the fields that are based on the gradient of the coarse graining function.

$$\mathbf{A}(\mathbf{r}) = \sum_i A_i \otimes \nabla\phi(\mathbf{r}_i - \mathbf{r}) \tag{87}$$

It will be used as a step in calculating the deformation and deformation rate, defined in equations 52 and 53.

The structure of the algorithm is very similar to the normal coarse graining algorithm, with only minor changes to the the weight Φ. As the gradient of ϕ, it now becomes a vectorial quantity $\mathbf{\Phi}$ and thus the fields will acquire an additional dimension over their corresponding particle properties. The multiplication of the weight and property is replaced by the outer product of tensors. The gradient $\nabla\phi$ depends on the choice of the coarse graining function ϕ.

```
 1  for($i=0$ to $i=N-1$)
 2  {
 3      $\boldsymbol{\lambda}_- =$ int$((\mathbf{r}_i - \mathbf{c})/\mathbf{g})$
 4      $\boldsymbol{\lambda}_+ =$ int$((\mathbf{r}_i + \mathbf{c})/\mathbf{g})$
 5      for($\boldsymbol{\lambda} = \boldsymbol{\lambda}_-$ to $\boldsymbol{\lambda} = \boldsymbol{\lambda}_+$)
 6      {
 7          $\mathbf{\Phi} = \nabla\phi\left(\widetilde{\mathbf{r}}_{\boldsymbol{\lambda}} - \mathbf{r}_i\right)$
 8          for($k=0$ to $k=M-1$)
 9          {
10              $\widetilde{A}^k_{\boldsymbol{\lambda}} \mathrel{+}= A^k_i \otimes \mathbf{\Phi}$
11          }
12      }
13  }
```

Algorithm 2 – The modified coarse graining algorithm for the fields that depend on the gradient $\nabla\phi$. The algorithms are identitcal except for lines 7 and 10.

Being almost identical to the normal coarse graining algorithm of the previous section, the computational complexity is also the same.

$$\mathcal{O}\left(NM\right) \tag{88}$$

Although it should be noted, that if the number of spatial dimensions d was a variable, it would also appear as a linear factor. The gradient adds a dimension with the size of d to the resulting output fields. This is irrelevant in granular media, but coarse graining has applications beyond this topic.

3.2.3 Mean Value Discretization of Particles as Cubes

The final algorithm will implement the mean value discretization, specifically equation 31.

$$\widetilde{A}_j = \frac{1}{V}\sum_i \frac{A_i}{U_i}\int_{C_j\cap Q_i} d\mathbf{r} \tag{89}$$

The components of $\mathbf{c}$ are now the dimensions of the approximation cuboid Q_i.

```
 1  for($i=0$ to $i=N-1$)
 2  {
 3      $\boldsymbol{\lambda}_- =$ int$((\mathbf{r}_i - \mathbf{c})/\mathbf{g})$
 4      $\boldsymbol{\lambda}_+ =$ int$((\mathbf{r}_i + \mathbf{c})/\mathbf{g})$
 5      for($\boldsymbol{\lambda} = \boldsymbol{\lambda}_-$ to $\boldsymbol{\lambda} = \boldsymbol{\lambda}_+$)
 6      {
 7          $\Phi = \left(\int_{C_{\boldsymbol{\lambda}}\cap Q_i} d\mathbf{r}\right)/(VU_i)$
 8          for($k=0$ to $k=M-1$)
 9          {
10              $\widetilde{A}^k_{\boldsymbol{\lambda}} \mathrel{+}= A^k_i \Phi$
11          }
12      }
13  }
```

Algorithm 3 – The algorithm for mean value discretization. The second loop ensures that λ only iterates over cells that are known to be intersected by the cuboid Q_i. The integral is then easily evaluated with equation 29. Apart from the calculation of the weight Φ, this algorithm is identical to the standard coarse graining algorithm.

Here, as opposed to the other coarse graining algorithms, the cutoff $\mathbf{c}$ is not linked to the cell size. In other words: The size of the particle and the cell are independent of each other. This causes the computational complexity of this algorithm to scale with the total number of cells G, if the grid spacing is smaller than the particles.

$$\mathcal{O}(NMG) \tag{90}$$

While this might seem like a disadvantage at first, it is not. The other coarse graining algorithms can only meaningfully be used with a grid spacing similar to the size of the particles, so G is not a free variable. However, with this mean value discretization the resolution can be increased arbitrarily to more closely approximate the true underlying field.

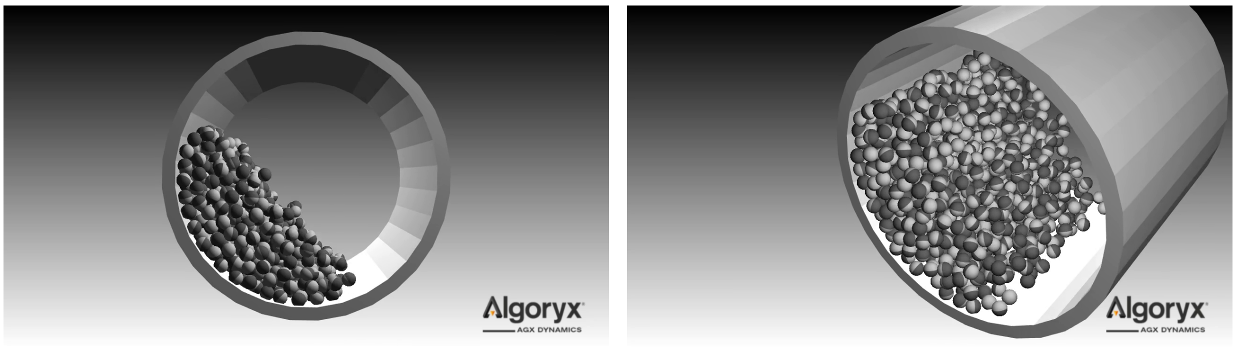
4 Simulation of a Rotating Drum

A rotating drum, containing a number of particles, is used as a test case on which we verify the algorithms. Some of the coarse grained fields, defined in section 2.5.1, will be plotted to validate their qualitative correctness. Additionally, the problem of non-conservation of the total quantities in the coarse graining process, as described in section 2.2.3, will be demonstrated. We will also show that mean value discretization, as defined in section 2.3, fixes this issue. Furthermore, we will use this simulation while developing a data driven model to predict particle motion in the last parts of this thesis.

4.1 The Scene

The drum itself is not perfectly circular but consists of 32 panels that approximate a circle with an inner radius of 1m. The front and back wall consist of the same material and rotate along with it, but are not rendered in the figures. Particles of identical size are spawned within the drum in a regular lattice (with a small random jitter), consisting of $10^3 = 1000$ particles. The particles have a radius of approximately 0.06m. The entire scene is shown in figure 8.

The drum is rotating at a constant speed with no regard to any interaction with the particles. The time step is $\Delta t = (1/60)\,\mathrm{s}$.



(a) Frontal view. **(b)** View into the drum.

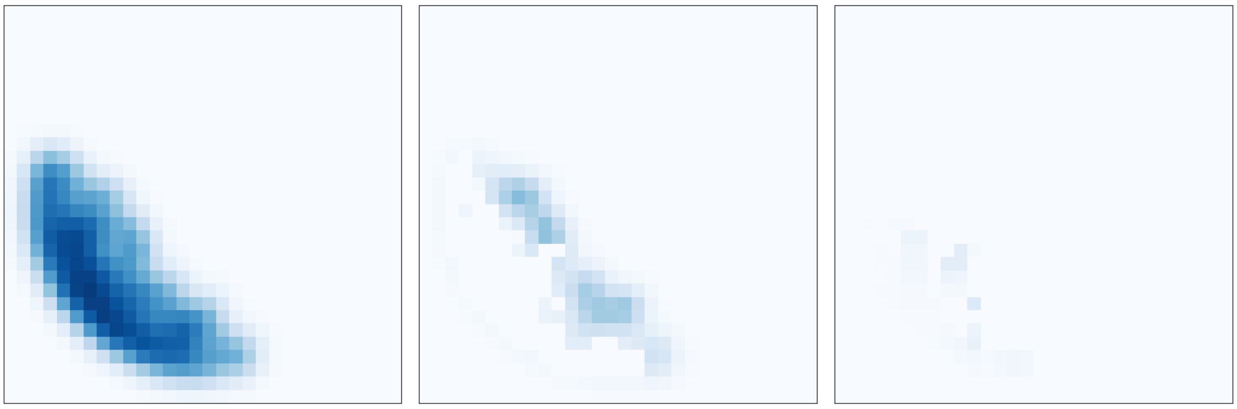
Figure 8 – The scene, consisting of a rotating drum that contains particles, shown from two perspectives.

The coarse graining region extents slightly beyond the drum in every direction. It is divided into $30^3 = 27000$ identical cubic cells. The coarse graining width w is chosen to equal the size of one such cell at about 0.08m.

The coordinate system is oriented such that, in figure 8a, the x-axis points to the right, the y-axis points into the plane, and the z-axis points up.

4.2 The Fields

Slices of select coarse grained fields are shown in figures 9 to 12. The slices are taken from the center of the drum. The images all show the same point in time, at which some components seem to disappear. Blue colors represent positive values, while red colors represent negative values. However, colorscales are omitted because we are only interested in the relative distribution throughout the drum.



(a) Mass **(b)** Pressure **(c)** Stress von Mises

Figure 9 – The scalar fields.

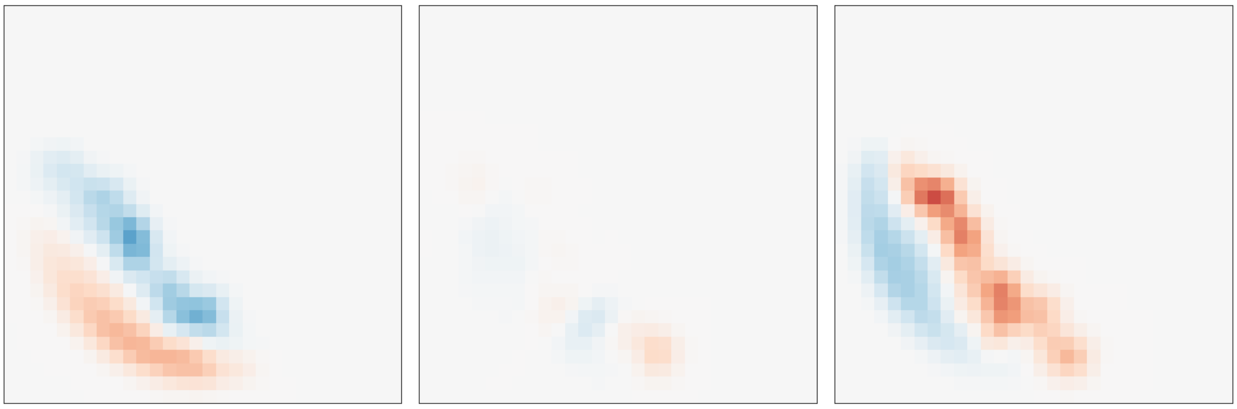


Figure 10 – The three components of the momentum field.

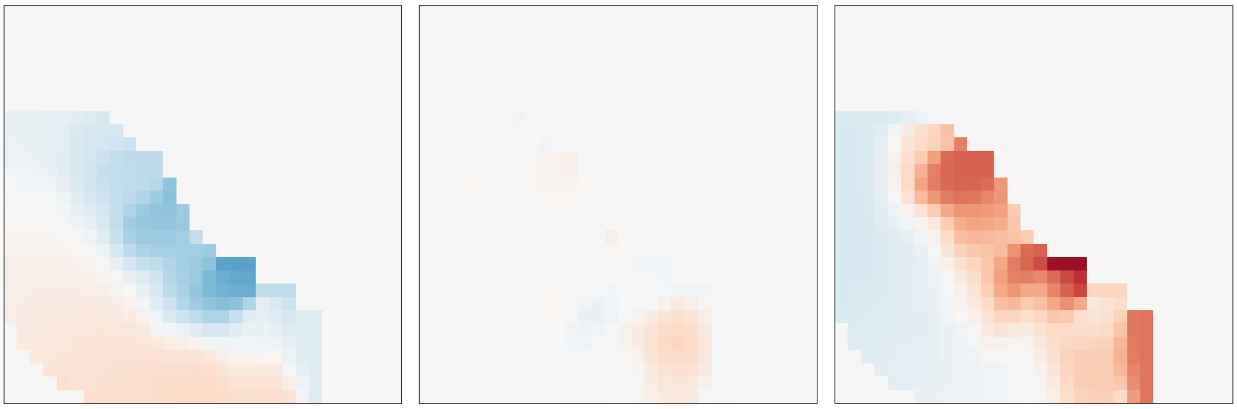


Figure 11 – The three components of the velocity field.

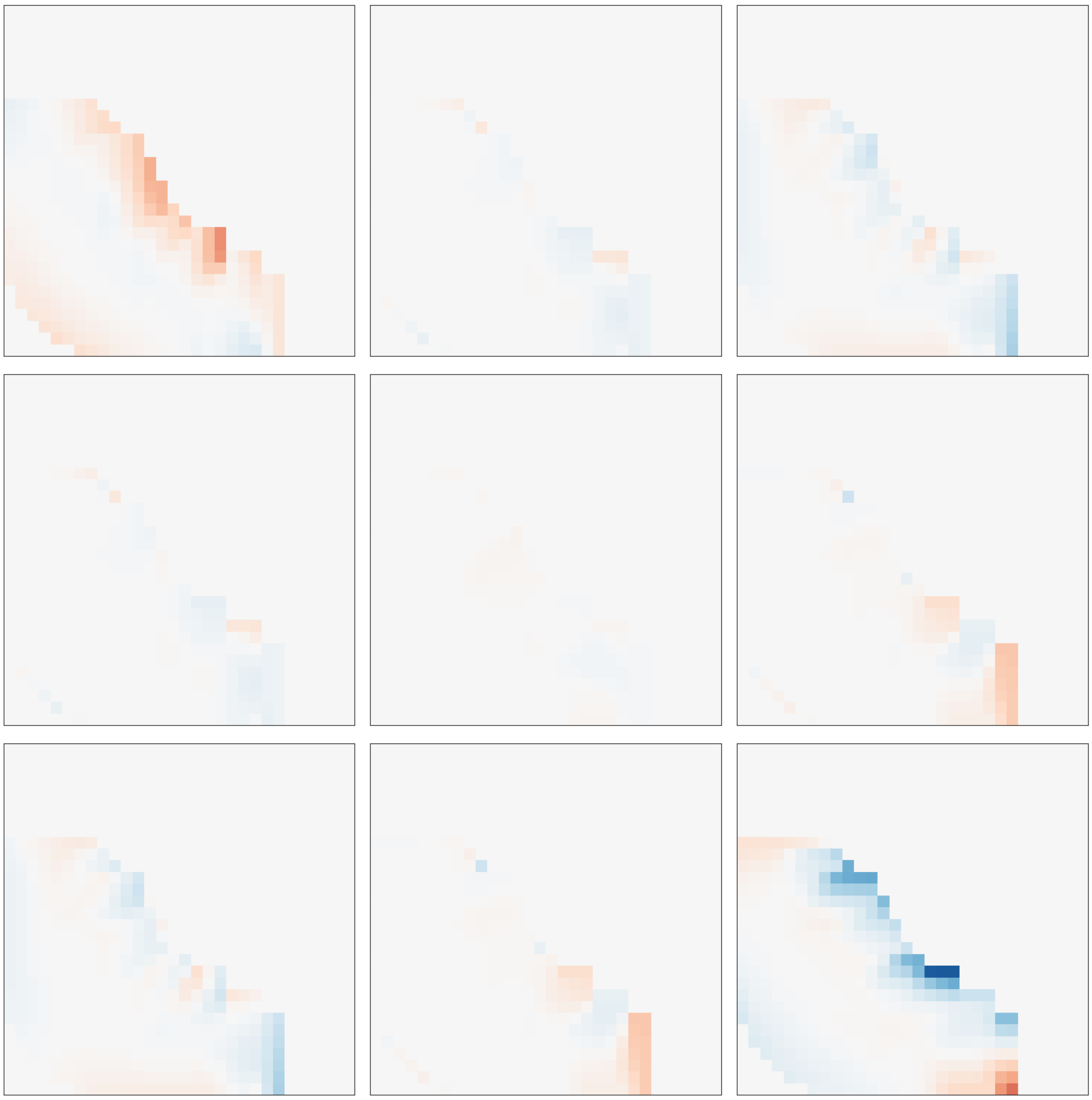


Figure 12 – The nine components of the strain rate field.

4.3 Non-Conservation of Total Quantities

As described in section 2.2.3, the total quantites of the particles are, in theory, not conserved in the coarse graining process. To investigate this in practice, the total value of the coarse grained mass field in this simulation is plotted over time and compared to the total mass of the particles, which is a constant. The same is done for the mean value discretization alternative, as defined in section 2.3, which we claim fixes the issue. This is shown in figure 13.

As expected, the mass of the coarse grained field fluctuates, although the magnitude of the fluctuations is fairly small. More significant is the consistently lower total mass. This can likely be attributed to the finite cutoff, as defined in section 3.2.1. The fluctuation in total mass is representative of all particle properties.

The total mass of the direct discretization approach is almost identical to the total mass of the particles. The minute differences here can be attributed to finite machine precision.

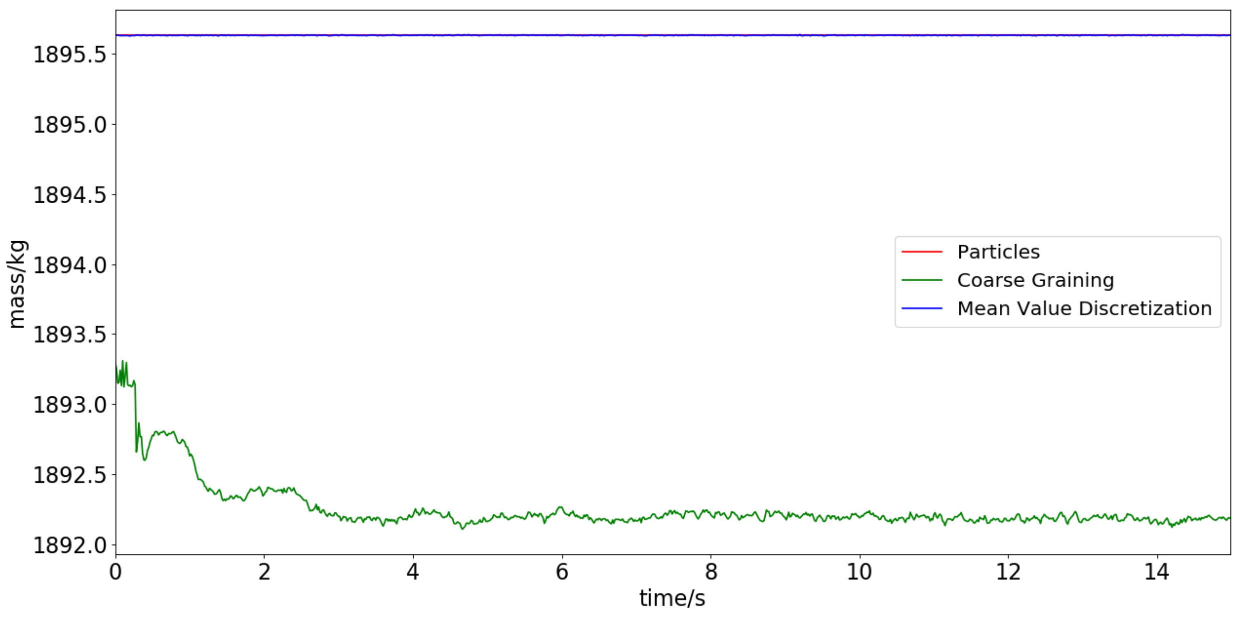


Figure 13 – The total mass of the particles and the fields in the rotating drum. The red line is hardly visible as it is almost completely covered by the blue line.

5 Data Driven Model

The position of any given particle $\mathbf{r}\,(t+\Delta t)$ after a time step Δt can always be written as its current position $\mathbf{r}\,(t)$ plus a displacement $\Delta\mathbf{r}$.

$$\mathbf{r}\,(t+\Delta t) = \mathbf{r}\,(t) + \Delta\mathbf{r}\,(t,\Delta t) \tag{91}$$

This can be approximated to first order, using the velocity $\mathbf{v}$ at time t [21].

$$\mathbf{r}\,(t+\Delta t) = \mathbf{r}\,(t) + \mathbf{v}\,(t)\,\Delta t + \mathcal{O}\left(\Delta t^2\right) \tag{92}$$

For uniform motion, i.e. when there is no acceleration, the nonlinear term vanishes and the new position of the particle can be calculated entirely from its current position and velocity. These quantities are known at every time step in the simulation, using the *AGX* physics engine.

Because the physics engine works with discrete time steps of a fixed duration, we will modify the notation slightly. The plain symbols $\mathbf{r}$ and $\mathbf{v}$ will denote the current position and velocity, while the primed $\mathbf{r}'$ will denote the position after a single time step. Additionally we introduce the symbol $\mathbf{K}$ to represent the nonlinear term.

$$\mathbf{r}' = \mathbf{r} + \mathbf{v}\,\Delta t + \mathbf{K} \tag{93}$$

We call this term $\mathbf{K}$ the *displacement deviator*, as it is the deviation of the displacement $\mathbf{r}' - \mathbf{r}$ from that of uniform motion $\mathbf{v}\,\Delta t$. The goal for the data driven model is to predict $\mathbf{K}$ without actually solving the physics of the system.

5.1 Input and Output Data

The idea is to use a particle's intrinsic properties and the coarse grained field data in the particle's surroundings to predict the nonlinear term. For this purpose the field data of the cell containing the particle as well as the 26 adjacent cells (a 3x3x3 block) is assembled into a *feature* vector, which serves as the input to a neural network.

Because the goal of the model is to learn a generic behaviour, we can not include a particle's position as input to the model. In that case the model could make simple inferences of how a particle tends to move in a certain location and would consequentially be tied to a specific simulation. Thus, we instead use the offset $\mathbf{s}$ of the particle from the nearest point of the grid. This restricts the feature vector to have only local information. Additionally we include the particle's velocity.

The coarse grained field are also included in the feature vector. Again, we only want to use local information, so only the values of the cell within which the particle is located, as well as the 26 surrounding cells are included. In total that gives a feature vector consisting of the following data:

▶ Particle Offset	$\mathbf{s}^{\mathrm{p}}$
▶ Particle Velocity	$\mathbf{v}^{\mathrm{p}}$
▶ Mass	m
▶ Pressure	P
▶ Stress Von Mises	σ^{vm}
▶ Momentum	$\mathbf{p}$
▶ Velocity	$\mathbf{v}$
▶ Local Force	$\mathbf{f}$
▶ Force	$\mathbf{F}$
▶ Stress	$\boldsymbol{\sigma}$
▶ Strain Rate	$\boldsymbol{\varepsilon}'$

To ensure stable training, each particle property and field is shifted by the corresponding mean value in the dataset and divided by its standard deviation[22]. This results in a distribution with a mean of 0 and a standard deviation of 1. These mean values and standard deviations are saved alongside the model, to ensure that the same procedure can be applied to future data that is not part of the training dataset. The normalized collection of all the inputs is called the *feature* vector $\mathbf{x}$. At this point it feeds directly into the neural network.

The output data consists of only the three components of the *displacement deviator* $\mathbf{K}$. These components are normalized in the same manner as the input data. The normalized output is called the *label* vector $\mathbf{y}$. It is precisely this, that the model learns to predict. This is then converted to the value of $\mathbf{K}$ by multiplying with the previously saved standard deviation and adding the previously saved mean.

5.2 Training

The model makes use of supervised training; It uses a precomputed dataset that provides *samples* in the form of pairs of the *feature* vector and the *label* vector. This dataset is split into two parts: One for training the model and one for validating its performance. Roughly 80% of samples are used for training and 20% for validation.

5.2.1 The Loss Function

The loss function $\mathcal{L}$ is used to train the model. It is formulated as the mean square error of the predicted labels from the true labels. The *label* vector will be denoted as $\mathbf{y}^{\mathrm{T}}$ for the true labels in the dataset and as $\mathbf{y}^{\mathrm{P}}$ for the prediction of the model. The loss function is then:

$$\mathcal{L} = \frac{1}{3} \sum_{\mu} \left(\left(y^{\mathrm{P}}\right)^{\mu} - \left(y^{\mathrm{T}}\right)^{\mu} \right)^2 \tag{94}$$

5.2.2 The Validation Metric

The loss function makes use of the normalized data instead of the plain *displacement deviator* for greater numerical stability during training. The real quantity of interest, however, is the error of the plain *displacement deviator* $\mathbf{K}$. Hence, the mean square error E_i of the *displacement deviator* is the quantity by which we actually judge the performance of the model for a given sample i in the validation dataset.

$$E_i = \frac{1}{3}\sum_\mu \left((K^{\mathrm{P}})^\mu_i - (K^{\mathrm{T}})^\mu_i \right)^2 \tag{95}$$

This quantity is defined per particle. To calculate a single number for the whole validation dataset we simply compute the mean E over all samples within it.

$$E = \frac{1}{N}\sum_i E_i \tag{96}$$

5.2.3 Judging the Performance

To judge the performance of the model, one needs to compare it to something. The obvious choice for a comparison baseline is the prediction that uniform motion would make. By virtue of construction (see equation 93), this prediction is precisely $\mathbf{K}^{\mathrm{U}} = \mathbf{0}$.

Using this, a quantity per particle can be defined analogously to the validation metric in equation 95.

$$E^{\mathrm{U}}_i = \frac{1}{3}\sum_\mu \left((K^{\mathrm{U}})^\mu_i - (K^{\mathrm{T}})^\mu_i \right)^2 = \frac{1}{3}\sum_\mu \left((K^{\mathrm{T}})^\mu_i \right)^2 \tag{97}$$

Like before, this gives an overall error for the entire validation dataset.

$$E^{\mathrm{U}} = \frac{1}{N}\sum_i E^{\mathrm{U}}_i \tag{98}$$

The quotient of the validation metric E and this baseline value E^{U} can now be calculated to give a single number η to judge the performance of the model.

$$\eta = \frac{E}{E^{\mathrm{U}}} \tag{99}$$

The smaller this number, the better the prediction; A number $\eta < 1$ directly corresponds to the prediction of the model being better than the prediction of uniform motion.

5.2.4 The Optimizer

The *Adam*[20] optimizer was used for training the network. In initial test cases to decide on an optimizer it resulted in faster training, compared to stochastical gradient descent. It achieved this without compromising quality of the fully trained model. Also, it required less manual tweaking of training parameters, such as the learning rate and batch size.

5.3 The Model

Predicting the *labels* from the *features* is a typical regression problem. We employ a multilayer perceptron[23] to perform this regression.

The final model is shown in figure 14. It is the result of manual tweaking of the number of hidden layers and the number of nodes in each layer to balance training performance and resulting validation loss. More layers tended to result in longer training times with diminishing improvements, while less layers simply had worse validation losses. While it performs satisfactory, no claims are made that this model is optimal. *Hyperparameter optimization*[24] could likely be employed to improve it further.

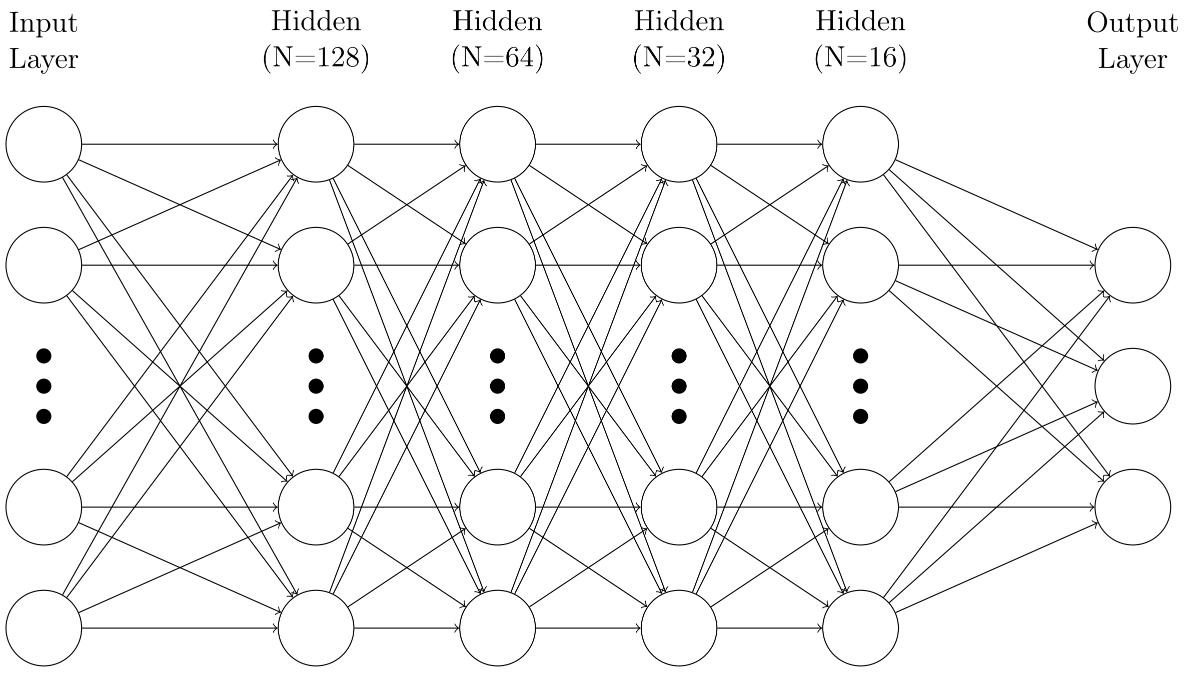


Figure 14 – The model to predict the displacement deviator. All of its layers are densely connected, with four of it being hidden layers of decreasing size.

In the following section, the number of fields included in the feature vector varies. The input layer always takes the size of the feature vector, so the number of nodes in this layer varies along with the included fields.

The output layer corresponds to the displacement deviator, so it has a fixed size of 3 nodes.

5.4 Relevance of The Input Fields

Several models were trained, that vary only in the fields included in the *feature* vector. For each of these cases the number η, defined in equation 99, was calculated. To mitigate fluctuations caused by the statistical nature of the training, the entire process was repeated 10 times and the mean of the η values was calculated.

Fields	None	All	m	P	σ^{vm}	$\mathbf{p}$	$\mathbf{v}$	$\mathbf{f}$	$\mathbf{F}$	$\boldsymbol{\sigma}$	ε'
η	0.86	0.65	0.66	0.84	0.84	0.69	0.64	0.84	0.85	0.85	0.66

Table 1 – The means of the η values over 10 runs, when including different fields in the *feature* vector. The standard deviation of each of these values is 0.01.

These results clearly show two distinct levels for η. One is around $\eta = 0.85$ and one is around $\eta = 0.65$.

When including no fields at all in the *feature* vector, the performance number is around this first level of $\eta = 0.85$. (Note that the *feature* vector still contains the particle's offset and velocity.) This value being smaller than 1 means that using only a particle's intrinsic properties allows for a better prediction of the *displacement deviator* than simply assuming uniform motion. This is not surprising, because in this specific simulation certain inferences can be made with the particle's intrinsic data. For example, particles with velocity beyond a certain threshold will tend to slow down. The only way to accquire high velocities is when they are rolling/falling down the pile and they will inevitably be stopped by the drum.

The second level at around $\eta = 0.65$ is more interesting, because this gives a tangible improvement that stems from the additional information that the fields provide. The fields that produce these improvements are the mass m, pressure P, momentum $\mathbf{p}$, velocity $\mathbf{v}$ and strain rate $\boldsymbol{\varepsilon}'$. However, when including all the fields simultaneously there is no further improvement. This is likely due to a lot of the information used to calculate these fields being redundant.

5.5 Source of Errors

While the previous section shows a successful application of the model, the obvious question is now: Where does the remaining error come from?

To possibly find an answer to this question, we render the scene with the colors of the particles mapped to the corresponding prediction errors (as defined in equation 95). This is shown in figure 15 and should give us an idea of which particles the largest errors originate from.

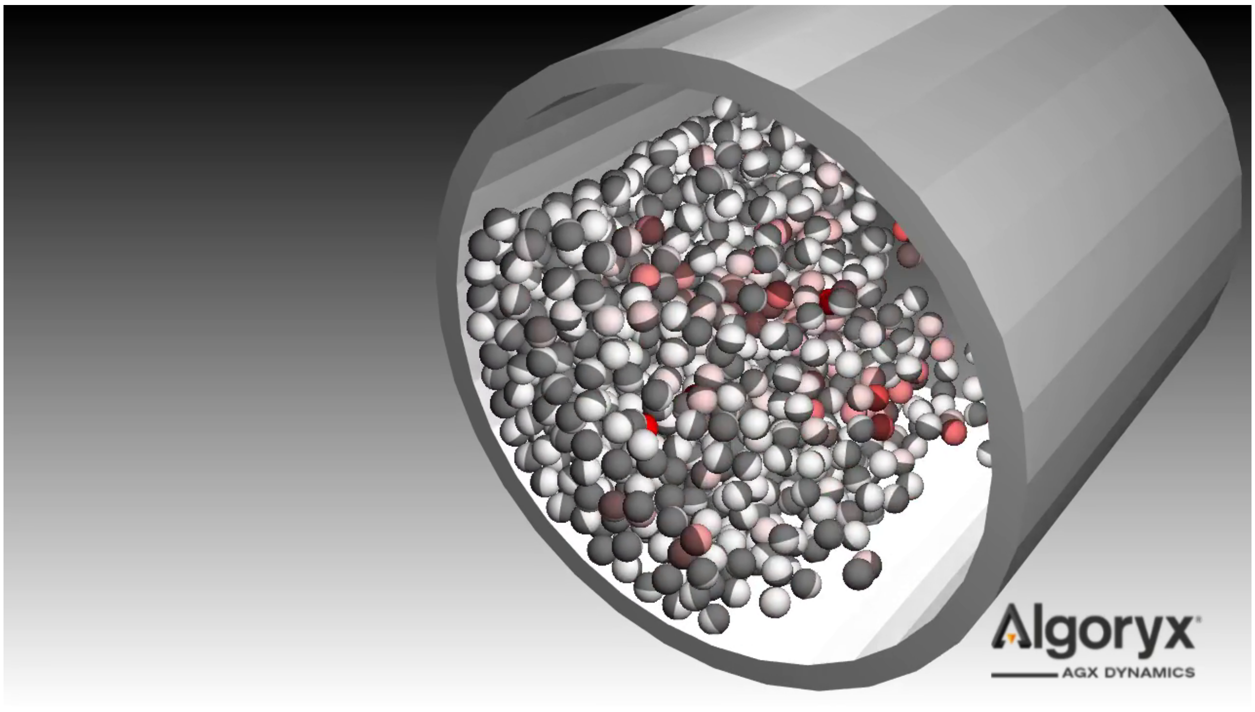


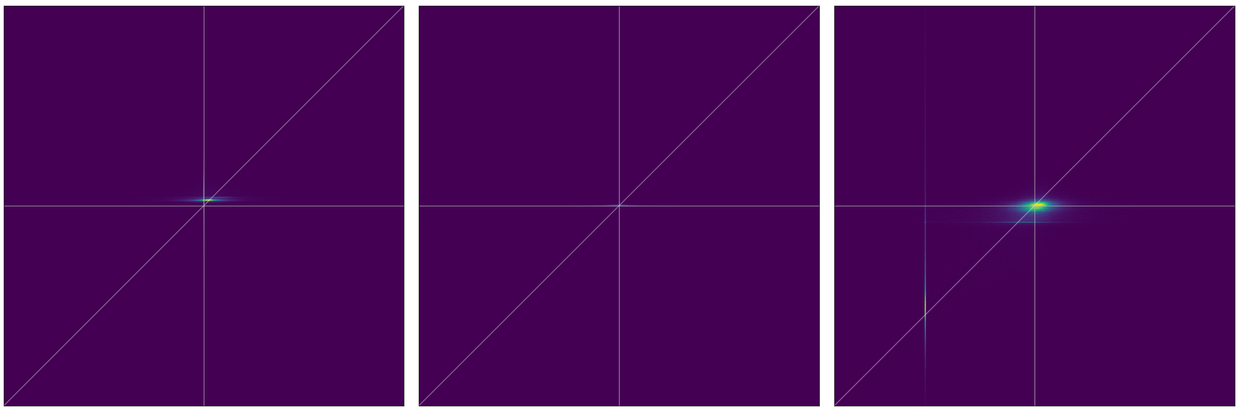
Figure 15 – This figure shows the error as defined in equation 95. It is calculated from the results of a pretrained model that is applied to a similar scene.

It is difficult to make any precise statements about the particles in this figure. The errors seem to be most prevalent among the particles that are part of the surface flow. Large errors seem to occur especially where the particles hit the drum and are abruptly stopped. This demonstrates a key limitation of the model. The drum itself is completely inexistant in the particle and field data. It is thus not surprising that we find some of the largest errors here.

The true displacement deviator is directly related to a particle's average acceleration $\bar{\mathbf{a}}$ over the time step[21].

$$\mathbf{K}^{\mathrm{T}} = \frac{1}{2}\bar{\mathbf{a}}\,\Delta t^2 \tag{100}$$

This average acceleration is, of course, as unknown as the displacement deviator itself, but it can give a more intuitive understanding of the error. When the particles hit the drum, they experience a large acceleration by colliding with an object that is invisible to the model.



(a) x component **(b)** y component **(c)** z component

Figure 16 – A heatmap of the predicted displacement deviator $\mathbf{K}^{\mathrm{P}}$ on the y-axis vs. the true displacement deviator $\mathbf{K}^{\mathrm{T}}$ on the x-axis. The units are arbitrary, but the scalings of the plots are identitcal. The diagonal line shows the identity, which represents the ideal situation in which the predictions are perfect.

Plotting the predicted displacement deviator $\mathbf{K}^{\mathrm{P}}$ against the true displacement deviator $\mathbf{K}^{\mathrm{T}}$ in figure 16 allows us to gain at least a bit of insight into where the error originates. First of all, there is a statistical spread to the data that contributes to the error. The prediction of uniform motion, for comparison, would simply project every datapoint onto the x-axes of the plots. Consequently, a distribution closer to the diagonal than the x-axis corresponds directly to the model prediction being better than the uniform motion prediction and vice versa.

However, of more interest are the more distinct features, such as the vertical line of datapoints in the z component. The data along this line likely represents free-falling particles, as their displacement deviator is expected to be 0 in the x and y direction but a negative constant in the z direction. This constant is, of course, related the the gravitational acceleration by equation 100. While the distribution for these predictions is skewed towards the diagonal, there is still a relatively large spread to them.

Also of note is that the spread of the distribution in the z component is much larger than the other components. The spread in the y component in particular almost vanishes in comparison, whereas the x component forms a thin horizontal band. This corresponds to a small variation in the predictions compared to the variation of the true displacement deviators. The same is found, to a lesser extent, in the z component. This means, that the model makes rather conservative predictions and thus tends to underestimate the magnitude of the true displacement deviator.

Lastly, we will take a look at the overall error distribution. For that, we bin the relative frequency $P(E_i)$ of the prediction errors E_i and plot them in a histogram in figure 17.

The histogram roughly follows a line on the log-log scale, suggesting that the errors occur with a frequency of $P(E_i) \approx a(E_i)^b$, with a and b being some parameters. A rough estimate puts the value for b, which corresponds to the slope in the histogram, at a value of around $-4/3$.

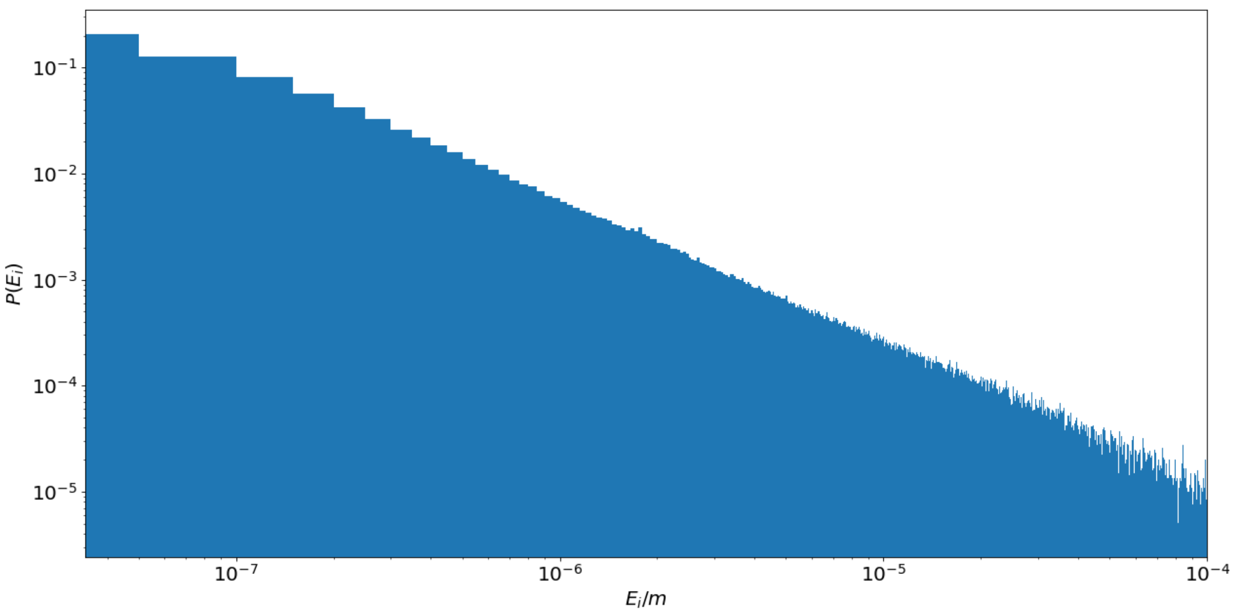


Figure 17 – The relative frequency of the mean square errors E_i in a histogram on a log-log scale. The values are normalized, such that the sum of all bins equals 1.

6 Conclusion

Coarse graining provides a method for transitioning from a discrete element description of a system to a continuum description. It has previously been shown analytically that the coarse graining process gives rise to the expected equations of motion of the continuum[3]. However, we show that in numerical simulations the total contribution of particles to the field becomes position dependent. Additionally, it was shown that coarse graining discards any information about particle size.

An alternative to coarse graining was proposed in section 2.3. It was shown that the information of the particle size is not lost. This allows for the discretization of particles with widely varying sizes in a grid of an arbitrary resolution. This alternative approach also conserves the total particle properties (e.g. mass) and removes any position dependence for them. However, a disadvantage is that the fields obtained in this manner were not yet shown to strictly obey other equations of motion. The definitions for these fields were merely chosen to mimic those of standard coarse graining.

This might pose an opportunity for future theoretical work to develop a coarse graining procedure that adresses these issues. Ideally it would combine the advantages of both methods, allowing for coarse graining of arbitrarily sized particles in a grid of an arbitrary resolution while obeying the continuum equations of motion.

Both procedures were developed and implemented as algorithms, which were then tested on a simulation of a rotating drum, containing a large number of particles. This scene, along with the coarse grained fields, was then used as the basis for a machine learning model. It was possible to demonstrate a tangible improvement in the prediction of particle movement when the local fields were used in the model as opposed to when they were not. While the predictions are still of rather poor quality, the true benefit of this model is, in theory, the potential for generalization. Previous works have made more successful predictions using other methods[25], but these models are only trained for a predetermined scene. The model in this thesis, however, uses only local data to predict a particle's movement, so it might be possible to train the model in a training scene and then apply it to very different scenes.

However, a problem that arises with neural networks in general is that they are usually treated as black boxes. Among other issues, this makes it inherently difficult to interpret the errors[26]. The model developed in this thesis is no exception; while the error was quantified, it is difficult to explain.

A possible subject for the future would be to take a step back and try to replicate the results of the neural network with a statistical model[14], while still using only local field data. If a similar result could be produced with the statistical model, it would provide more insight into the source of errors. This knowledge might then be helpful to improve the accuracy of the entire model.

Another prospect for possible future work is the investigation of the effects other fields might have on the predictions. The granular temperature was omitted in the data driven model because it can not be calculated with the algorithms developed in this thesis. It is, however, a good candidate for improving the predictions, as it provides information on how much the particle velocities deviate from the mean within a given cell. A possible alternative, that can be calculated with the developed algorithms while providing similar information, is the standard deviation of velocities. Also, the kinetic and contact stress might be seperated, because, depending on the situation, one of them may overshadow the other. In fact, the kinetic stress might be removed altogether, because the information present in this field is very similar to the momentum and velocity fields. Lastly, one might come up with synthetic fields, derived from arbitrary combinations of particle properties, to try and improve predictions further. These synthetic fields need not have an intuitive physical interpretation.

References

[1] Bruno Andreotti, Yoël Forterre, and Olivier Pouliquen. *Granular Media: Between Fluid and Solid.* Cambridge University Press, 2013.

[2] Stefan Luding. Introduction to discrete element methods. *European Journal of Environmental and Civil Engineering*, 12(7-8):785–826, 2008.

[3] Isaac Goldhirsch. Stress, stress asymmetry and couple stress: from discrete particles to continuous fields. *Granular Matter*, 12:239–252, 5 2010.

[4] Jie Zhang, Robert P. Behringer, and Isaac Goldhirsch. Coarse-graining of a physical granular system. *Progress of Theoretical Physics Supplement*, 184:16–30, 03 2010.

[5] Juan Petit P., Juan Marín, and Leonardo Trujillo. On the construction of a continuous theory for granular flows. pages 463–471, 01 2014.

[6] Martin Müser, Godehard Sutmann, and R. Winkler. *Hybrid Particle-Continuum Methods in Computational Material Physics.* 01 2013.

[7] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, USA, 1988.

[8] B.D. Jones and J.R. Williams. Fast computation of accurate sphere-cube intersection volume. *Engineering Computations*, pages 1204–1216, 01 2017.

[9] J.J. Hopfield. Artificial neural networks. *IEEE Circuits and Devices Magazine*, 4(5):3–10, 1988.

[10] Kenji Kawaguchi. Deep learning without poor local minima. 05 2016.

[11] Martin Anthony. *Discrete Mathematics of Neural Networks.* Society for Industrial and Applied Mathematics, 2001.

[12] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. *CoRR*, abs/1710.05941, 2017.

[13] J. Hertz, John, Krough, Anders Flisberg, Palmer, and Richard G. *Introduction To The Theory Of Neural Computation.* 12 1991.

[14] W. Sarle. Neural networks and statistical models. 1994.

[15] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, and Dario Amodei. Language models are few-shot learners. 05 2020.

[16] Cameron Buckner and James Garson. Connectionism. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy.* Metaphysics Research Lab, Stanford University, Fall 2018 edition, 2018.

[17] Shun ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4):185–196, 1993.

[18] Oleg Burdakov, Yuhong Dai, and Na Huang. Stabilized barzilai-borwein method. *Journal of Computational Mathematics*, 37(6):916–936, 2019.

[19] David Saad. *On-Line Learning in Neural Networks.* Publications of the Newton Institute. Cambridge University Press, 1999.

[20] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[21] R. Feynman, R. B. Leighton, and M. Sands. *The Feynman Lectures on Physics*. Basic Books, new millenium edition, 2011.

[22] Yann Lecun, Leon Bottou, Genevieve Orr, and Klaus-Robert Müller. Efficient backprop. 08 2000.

[23] Fionn Murtagh. Multilayer perceptrons for classification and regression. *Neurocomputing*, 2(5):183–197, 1991.

[24] Marc Claesen and Bart De Moor. Hyperparameter search in machine learning. 02 2015.

[25] Erik Wallin and Martin Servin. Data-driven model order reduction for granular media. *Computational Particle Mechanics*, 02 2021.

[26] Vanessa Buhrmester, David Münch, and Michael Arens. Analysis of explainers of black box deep neural networks for computer vision: A survey, 2019.