Minimal Packing Sets

Lorne Arnold

Abstract

Soil is fundamentally a discrete material that is, nevertheless, commonly modeled as a continuum in part because of the computational expense of large-scale discrete element models (DEMs). Even at lab specimen scales, DEM's computational cost may be substantial depending on the grain sizes being modeled. Despite these limitations, discrete models have proven useful in furthering our understanding of soil mechanics because they can spontaneously replicate realistic soil behavior as an emergent macro-scale property from a collection of particles following relatively simple interaction rules. This makes DEM an attractive tool for a multi-scale modeling approach where the constitutive behavior of a representative volume element (RVE) is characterized with a discrete model and applied at a larger scale through a continuum model. The ability of the RVE to represent a soil depends strongly on an appropriate grain size distribution match. However, in order to achieve computationally feasible models, even at lab scales, DEM simulations often use larger minimum particle sizes and more uniform distributions than their intended targets. Intuitively, discrete matches of different grain size distributions (GSDs) will require vastly different numbers of particles. But the precise relationship between GSD characteristics and the number of particles needed to match the distribution (and by extension the associated computational cost associated) is not intuitive. In this paper, we present the minimal packing set (MPS) concept. The minimal packing set is the smallest set of discrete particles needed to match a given GSD. We present a method for determining the MPS for any GSD and discuss strategies for finding the smallest MPS within a set of tolerances on the GSD. A mapping of USCS classification and MPS reveals a broad distribution of computational cost over several orders of magnitude for granular soils.

Introduction

```
import sys
sys.path.append('..')
import math
from gsd_lib import GSD, MinimalPackingGenerator
```

```python
import numpy as np
import scipy.stats as stats
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

plt.rcParams.update(
    {
        # Figure settings
        "figure.figsize": (5, 4),
        "figure.dpi": 400,
        "savefig.dpi": 600,
        "savefig.bbox": "tight",
        "savefig.pad_inches": 0.1,
        # Font settings
        "font.size": 12,
        "text.usetex": True,
        "font.family": "serif",
        "font.sans-serif": ["cmss10", "Arial"],
        # "font.serif": ["Latin Modern", "cmr10"],
        # "font.monospace": ["cmtt10"],
        "axes.formatter.use_mathtext": True,
        "axes.titlesize": 14,
        "axes.labelsize": 12,
        "xtick.labelsize": 10,
        "ytick.labelsize": 10,
        "legend.fontsize": 10,
        # Axes settings
        "axes.linewidth": 1.0,
        "axes.spines.top": True,
        "axes.spines.right": True,
        "axes.grid": True,
        "grid.alpha": 0.3,
        "grid.linewidth": 0.5,
        # Line and marker settings
        "lines.linewidth": 1.5,
        "lines.markersize": 6,
        "scatter.marker": "o",
        # Legend settings
        "legend.frameon": True,
        "legend.framealpha": 0.8,
        "legend.fancybox": True,
```

```
        "legend.numpoints": 1,
        # Tick settings
        "xtick.direction": "in",
        "ytick.direction": "in",
        "xtick.major.size": 4,
        "ytick.major.size": 4,
        "xtick.minor.size": 2,
        "ytick.minor.size": 2,
    }
)
```

Soil is a fundamentally discrete granular material whose complex mechanical behavior emerges from the numerous interactions between its constituent parts. The discrete element method (DEM), introduced by Cundall and Strack (1979), has proven to be a powerful tool in exploring the inter-particle interactions of granular assemblies. With DEM, a vast parametric space exists where grain sizes, shapes, contact models, etc. can be systematically varied and their influences on macro-scale assembly behavior can be quantified.

These models pose several challenges that must be managed in order for their benefits to be realized. Due to the number of elements involved in typical DEM simulations, both the computational resources needed to run them and the ability to characterize and interpret them are non-trivial. Over the years, the number of particles used in DEM simulations has increased, but not proportionally to the reduction in costs of computational resources. O'Sullivan (2014) showed that the average number of particles in DEM simulations increased substantially from 1998 to 2014, from approx. 10^3 to 10^5, but fell far short of the 10^7 predicted by Cundall (2001) for "easy" geomechanics problems by 2011. More recent publications in DEM have used particle numbers on the order of 10^6 (Sufian et al. (2021), Dong, Yan, and Cui (2022)) and in limited cases on the order of 10^8 (Fang et al. (2021), Zhang et al. (2024)).

To some extent, the number of particles used to study geomechanics may simply be limited by the number of particles needed to capture the behavior of interest. For example, studies have shown that for certain combinations of grain size distribution (GSD) and density, there may be significant portions of the soil mass that are not mechanically engaged with the soil matrix. This may lead to thresholds of particle size that can be omitted from a given simulation without significant impact on the behavior being studied. This depends, of course, on what the behavior of interest is. DEM studies focusing on shearing resistance will have different needs than those focused on permeability, for example.

Regardless of the application, DEM models need to have a sufficient resolution of particles to create a representative volume element (RVE). Several nuances to the RVE concept exist, but broadly speaking, the RVE represents the smallest volume of material to exhibit statistically consistent macro-scale behavior as its source material. In physical laboratory experiments and discrete numerical experiments, study samples sufficiently large to behave as an RVE is critical to the broader applicability of the results.

The RVE depends on several factors including the GSD and the behavior of interest. Samples with larger maximum particle sizes will have larger RVEs. Depending on the GSD, loading conditions, and behavior of interest, the minimum particle size expected to participate mechanically in the soil matrix may be substantially smaller than the largest particle size. Intuitively then, with increasing GSD breadth, the number of particles needed in a DEM simulation of granular assemblies will also increase. The ratio of minimum and maximum particle sizes is, therefore, a significant and often reported parameter in describing DEM models. Often, matching the true GSD of interest (e.g., from a physical soil sample) is sacrificed for computational efficiency, for example by upscaling the target GSD (Zeraati-Shamsabadi & Sadrekarimi, 2025).

While the insights we gain from DEM simulations with upscaled or truncated GSDs are valuable, there are acknowledged limitations imposed during scaling. Several questions around the limitations of DEM scaling are addressed by individual studies. First among these questions is whether a RVE-scale model has been achieved. But another important question exists around the computational cost savings associated with DEM scaling. Framed another way, an important question in experimental design could be: How computationally expensive would an unscaled DEM simulation of the material of interest be?

A brute force approach to answering this question certainly exists: build an unscaled DEM model and track the time required to run a small simulation with it. The obvious disadvantage of this approach is that finding the answer may be computationally expensive itself. A preferred approach would be an analytical solution that provides the number of particles of various grain sizes needed to reproduce any given GSD.

This paper introduces such an approach through the minimal packing set (MPS) concept. The minimal packing set is the smallest set of discrete particles needed to match a given grain size distribution by volume. In order to characterize the MPS, discrete mathematical descriptions of grain size distributions and granular samples are presented as sets and the rules describing several relationships between the sets are defined. Using these definitions, a simple algorithm is shown to efficiently identify the MPS. The results show a broad range of MPS magnitudes over several GSDs of interest in geotechnical engineering. The MPS algorithm is implemented in an open-source Python module, available on GitHub.

What is a grain-size distribution? How can it be described mathematically? How precise does a GSD need to be and how would one even measure that?

Discrete definitions

Identifying the mimimal packing set requires rigorous mathematical definitions for particulate samples, S, and grain size distributions, G. Conceptually, each of these are collections of items (i.e., sets) with specific restrictions.

Sample definition

A sample, S is an indexed set describing its particles as pairs of size, X_S, and quantity, Q_S. X_S is an ordered (i.e., each entry is larger than the previous) set of unique values of positive real numbers. Q_s is an unordered set of positive integers related to X_S by a shared indexing set for the sample, I_S.

$$S = \{(X, Q) \in I_S\} \tag{1}$$

where:

$$X_S = \{x_i : i \in I_S\} \text{ with } x_1 < x_2 < \cdots < x_{n_S} \tag{2}$$

$$Q = \{q_i : i \in I_S\} \tag{3}$$

$$X_S \subset \mathbb{R}^+ \text{ and } Q \subset \mathbb{Z}^+ \tag{4}$$

Grain size distribution definition

The grain size distribution, G, has a similar structure. It is and indexed set describing size boundaries, X_G, and masses, M_G. Like X_S, X_G is an ordered set of unique values, however, it's domain also includes zero (representing the pan in a typical sieve analysis). Like Q_S, M_G shares an index (I_G) with the size set, but it's domain is not restricted to integers, only to non-negative real numbers. The values in M_G represent the masses retained on a sieve with opening size X_G.

$$G = \{(X_G, M_G) \in I_G\} \tag{5}$$

where:

$$X_G = \{x_j : j \in I_G\} \text{ where } x_1 < x_2 < \cdots < x_{n_G} \tag{6}$$

$$M = \{m_j : j \in I_G\} \tag{7}$$

$$X_G \subset \mathbb{R}^{non-neg} \; ; \; V \subset \mathbb{R}^{non-neg} \tag{8}$$

Comparison definition

The relationship between S and G needs to quantify the match between the two in several respects. These are presented here with G being used to *describe* S. In the context of DEM modeling, this may seem backward because typically in DEM modeling, a target GSD is defined first and a sample generated to match it. On the other hand, a discrete sample (either physical or numerical) exists on its own, whereas a GSD only has meaning when interpreted as a description of a sample. Therefore the relationship between S and G will be defined in terms of G's description of S. In a later section, the topic of finding an instance of S that *matches* a target G will be addressed.

The relationship between S and G is formalized in three conditions described as follows:

Condition 1: G is complete if and only if the final entry in M_G is zero:

$$G \text{ is complete } \Leftrightarrow m_{n_G} = 0 \tag{9}$$

This condition ensures that x_{nG} provides an upper bound to the sizes described by G.

Condition 2: G describes S (denoted $G \longrightarrow S$) if and only if Condition 1 is met and the its smallest size is smaller than any size in S and its largest size is larger than any size in S:

$$G \longrightarrow S \Leftrightarrow \text{ Cond1} \wedge \min(X_G) < \min(X_S) \wedge \max(X_G) \geq \max(X_S) \tag{10}$$

Condition 3: G describes S accurately if Condition 2 is met and the combined masses of all the particles with sizes between every pair of sizes in X_G is equal to the retained mass on the lower of the size pairs in G:

$$G \overset{\text{accurately}}{\longrightarrow} S \Leftrightarrow \text{ Cond2} \wedge \sum_{\substack{i \in I_S \\ x_j < x_i \leq x_{j+1}}} q_i \cdot f(x_i) = m_j \tag{11}$$

where $f(x)$ is a scaling function that converts size to mass. As indicated in the equation below, "between" assumes (something about whether equal means retained or not).

Condition 4: G describes S articulately if Condition 2 is met and for all consecutive pairs of sizes in X_S there exists at least one size in X_G between them.

$$G \overset{\text{articulately}}{\longrightarrow} S \Leftrightarrow \text{ Cond2} \wedge \forall (x_i, x_{i+1}) \in X_S \exists (x_i < x_j \in X_G \leq x_{i+1}) \tag{12}$$

Physical interpretation

In physical terms, some assumptions are required in order for S and G to be interpreted as a physical sample and sieve analysis (or a numerical version of the same). First, the sizes in X_S and X_G should be one-dimensional since sieve analysis is based on particle diameter. Next, the scaling function, $f(x)$ (with x being diameter) should be defined. For spherical particles with of uniform density (ρ), this is a straightforward definition:

$$f(x) = \frac{\rho\pi}{6}x^3$$

Note, however, that the use of this particular scaling function is not required in general. It is only necessary that $f(x)$ be some mapping of $x \mapsto m$.

Solving for the minimal packing set

For any given S, finding G that accurately describes S is no more complicated that following the grain size analysis procedure described in ASTM D6913 (D18 Committee, n.d.). The reverse, finding a solution for S that is accurately described by some target G, is non-trivial. One of the reasons is that unlike a grain size analysis, the sizes of the solution (X_S) are bounded, but not explicitly defined. Additionally, the goal for this procedure is not only to find any S that is accurately described by G, but the minimal packing set S_{min} that is accurately described by G.

As a starting point, assuming that G is an articulate description of S and selecting any valid values for X_S, an indexed set of mass ratios, Φ, and size ratios, Ξ can be defined:

$$\Phi = \{\frac{m_j}{m_{n_G-1}} : j \in I_G\}; \text{ with elements } \phi_i$$

$$\Xi = \{\frac{f(x_i)}{f(x_{n_S})} : i \in I_S\}; \text{ with elements } \xi_i$$

The ratio Φ describes the masses in G relative the the mass of the largest size retaining mass in G. The ratio Ξ describes the mass (through the scaling function $f(x)$) of each particle size to that of the largest particle size in S. Since Φ describes relative masses of each size and Ξ describes the relative mass per particle of each size, the relative number of particles of each size Υ for any S described by G can be defined by the quotient of each entry in Φ and Ξ:

$$\Upsilon = \{\frac{\phi_j}{\xi_{n_S}} : i \in I_S\}; \text{ with elements } \upsilon_i$$

The ratio Υ is directly proportional to the key target parameter, Q, describing the quantities of each particle size in S. Unfortunately, with the exception of v_{nS}, the entries in Υ are not guaranteed to be integers, which is a requirement for Q. Multiples of Υ can be used to find its smallest possible integer representation. If some level of tolerable error can be specified, each successive iteration of multiples of Y can be mapped to its closest integer representation and compared to the target G to assess its error (i.e., how far Cond3 is from being satisfied). If the initial assumed sizes are held constant, this iterative approach will find the minimum set that meets a given error tolerance for the selected sizes.

But since the sizes in X_S are only bounded by X_G and not explicitly defined, the opportunity exists to find a better selection of entries in X_S that will minimize Q_S. This approach is equivalent to dropping the articulate description (Cond3) requirement from the initial attempt. The best allowable sizes for X_S can be found using the following steps:

1) Find the quantity ratios associated with the minimum ($-$) and maximum ($+$) allowable sizes in all but the largest sizes in X_S (i.e. Y_- and Y_+).
2) Check whether an integer is contained between each entry in Y_- and Y_+.
3) If not, iteratively increase Y_- and Y_+ until the span between each entry contains an integer.
4) When an integer quantity exists between each entry in Y_- and Y_+, populate Q_S with these integers.
5) Invert the mass scaling function $f(x)$ on the ratio Φ / Q_S to identify the sizes in X_S.

This procedure produces appropriate match of X_S and Q_S needed to satisfy the requirements of the minimal packing set, S_{min}. The effectiveness of the spanning integer algorithm is shown in the convergence of acceptable samples by iteration compared to a fixed size solution in Figure X.

```
# Setup for generating example grain size distributions

## Standard U.S. sieve sizes (mm)
x1 = np.array(
    [
        0.0001,  # Pan
        0.075,  # Number 200 sieve separates coarse from fines
        0.15,
        0.3,
        0.6,
        1.18,
        2.36,
        4.75,  # Number 4 sieve separates sand from gravel
        9.5,
        19,
        25,
```

```
        37.5,
        50,
        63,
        75,  # 3-inch sieve (100% passing this per ASTM D2487-17 1.2)

    ]
)

sieve_sizes = x1.copy()
n_sieves = len(sieve_sizes)

## Generate random grain size distributions on the sieve set
def mass_dist(n_sieves, rng=None, exponent=4.0, extra_randomness = 0):
    if rng is None:
        rng = np.random.default_rng()
    base_dist = 1*np.ones(n_sieves)  + rng.random(n_sieves)
    lower = 1 + n_sieves // 4
    upper = 1 + 3 * n_sieves // 4
    center_idx = rng.integers(lower, upper)
    distances = np.abs(np.arange(n_sieves) - center_idx) / center_idx
    scale_factor = np.exp(-exponent * distances)
    dist = base_dist * scale_factor
    for i in range(extra_randomness):
        dist *= 1 + rng.random(n_sieves)   # Add some extra randomness
    return dist

## A random generator with a fixed seed for reproducibility
rng = np.random.default_rng(1)

tol = 1e-2

gsd_list = []
set_size_list = []
flex_list = []

## Generate a small batch of GSDs
x = x1.copy()
min_span = 5
for start in range(0, len(x) - min_span):
    for end in range(start + min_span, len(x)):
        new_x = x[start:end]
        n_sieves = len(new_x)
```

```
        for j in range(10):
            # Create a distributed set of retained masses
            mass = mass_dist(n_sieves, rng, exponent=2, extra_randomness=7)

            mass[-1] = 0.0  # Ensure last mass is zero
            g = GSD(sizes=new_x, masses=mass)

            # Create a flexible minimal packing for each GSD
            mps_f = MinimalPackingGenerator(
                g, x_n_factor=0.5, tol=tol, flex=True, density=1.0
            )

            # Create a set size minimal packing for each GSD
            mps_s = MinimalPackingGenerator(
                g, x_n_factor=0.5, tol=tol, flex=False, density=1.0
            )
            gsd_list.append(g)
            set_size_list.append(mps_s)
            flex_list.append(mps_f)
```

```
# grain size distribution plot
# plt.close('all')
fig, ax = plt.subplots()
for gsd in gsd_list:
    ax.plot(gsd.sizes[1:], 100*gsd.percent_passing[1:], color='k', alpha=0.2, linewidth=0.5)

ex_i = 39
example_gsd = gsd_list[ex_i]
ax.plot(example_gsd.sizes[1:], 100*example_gsd.percent_passing[1:], color='r', linewidth=2, l
ax.plot(
    [
        example_gsd.sizes[1],
        example_gsd.sizes[1],
        example_gsd.sizes[-1],
        example_gsd.sizes[-1],
    ],
    [
        100 * example_gsd.percent_passing[1],
        0.5,
        0.5,
        100 * example_gsd.percent_passing[-1],
```

```
    ],
    color="r",
    linewidth=2,
    label="Example Retained",
)

ax.plot(
    [
        0.075,
        example_gsd.sizes[-1],
        example_gsd.sizes[-1],
        0.075,
        0.075,
        example_gsd.sizes[-1],
    ],
    [0.5, 0.5, 100, 100 * example_gsd.percent_passing[1], 0, 0],
    color="b",
    linestyle=":",
    linewidth=2,
    label="Uniform Distribution",
)

ax.plot([11,40], [52,52], color='k', linewidth=1,zorder=4)
left, bottom, width, height = (5.5, 40, 80, 30)
rect = plt.Rectangle((left, bottom), width, height, fill=True, color="1", alpha=0.8,zorder=3)

arrowprops = dict(arrowstyle="->", color="k", lw=1)
bbox = dict(fc="1", ec="1", alpha=0.9)


ax.add_patch(rect)
ax.annotate(
    r"\textbf{Curvature Index:}",
    xy=(5, 65),
    xytext=(6.5, 65),
    fontsize=11,
    # bbox=bbox,
    color="k",
    ha="left",
    va="center",
)
bbox = dict(fc="1", ec="r", linewidth=2, alpha=1)
```

```
ax.annotate(
    r"Area",
    xy=(5, 65),
    xytext=(20, 58),
    fontsize=11,
    bbox=bbox,
    color="k",
    ha="center",
    va="center",
)
bbox = dict(fc="1", ec="b", linestyle=":", linewidth=2, alpha=1)
ax.annotate(
    r"Area",
    xy=(5, 65),
    xytext=(20, 46),
    fontsize=11,
    bbox=bbox,
    color="k",
    ha="center",
    va="center",
)

ax.set_xscale('log')
ax.grid()#(True, which='both', axis='both')
ax.set_xlabel('Particle Size (mm)')
ax.set_ylabel(r'Percent Passing (\%)')
ax.set_ylim(0, 100)
ax.set_xlim(0.055, 100)
```

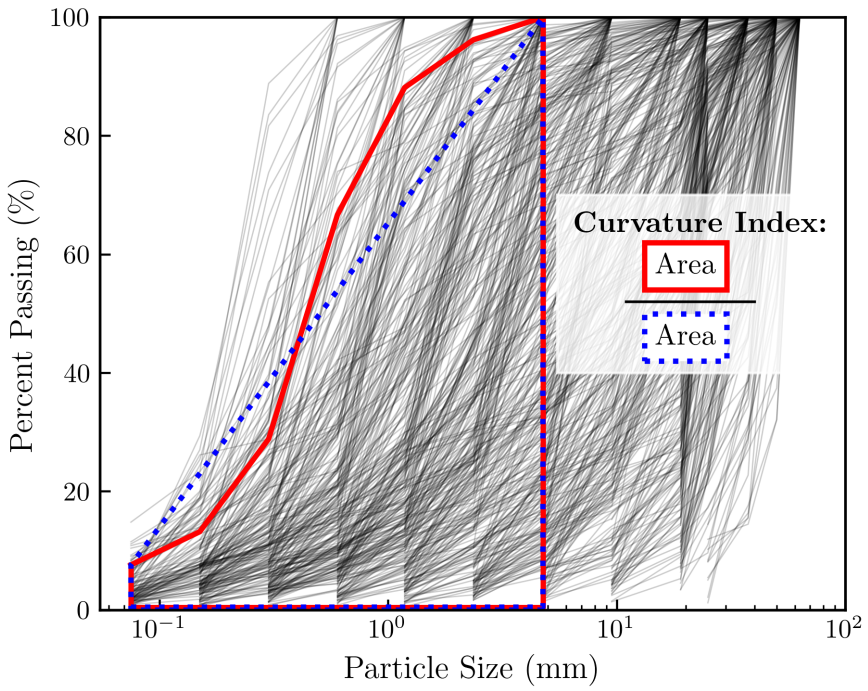

Figure 1: Grain size distributions evaluated. The area ratio defining the curvature index is shown. The example curve has a curvature index of 1.09.

```
# Packing algorithm convergence plot
fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True, height_ratios=[4, 1])
# fig, ax1 = plt.subplots()
j = 1
nf = []
ns = []

for i in range(len(set_size_list)):
    # f = flex_list[i]
    s = set_size_list[i]
    # xf = [n[-1] for n in f.qs]
    xs = [n[-1] - 1 for n in s.qs]
    # f_error = f.error
    s_error = s.error

    # nf.append(xf[-1])
    ns.append(xs[-1])
    # ax1.plot(xf, f.error, color="k", alpha=1)
    # ax2.plot(xf, f.error, color="k", alpha=0.5)
```

```
    ax1.plot(xs, s.error, color="r", alpha=0.5)

for i in range(len(flex_list)):
    f = flex_list[i]
    # s = set_size_list[i]
    xf = [n[-1] - 1 for n in f.qs]
    # xs = [n[-1] for n in s.qs]
    f_error = f.error
    # s_error = s.error

    nf.append(xf[-1])
    # ns.append(xs[-1])
    ax1.plot(xf, f.error, color='k', alpha=0.5)
    ax2.plot(xf, f.error, color="k", alpha=0.5)
    ax2.scatter(xf, f.error, marker='o',color='k', alpha=0.5)
    # ax1.plot(xs,s.error,color="r", alpha=1)


ax1.plot([0, 200], [tol, tol], color='k', linestyle='--', linewidth=1)



# # Calculate histograms
# counts_ns, bins_ns = np.histogram(ns, bins=50)
# counts_nf, bins_nf = np.histogram(nf, bins=bins_ns)

# # Normalize to max frequency
# norm_counts_ns = counts_ns / np.max(counts_ns)
# norm_counts_nf = counts_nf / np.max(counts_nf)

# # Plot normalized histograms
# ax2.bar(
#     bins_ns[:-1],
#     norm_counts_ns,
#     width=np.diff(bins_ns),
#     alpha=1,
#     label="Fixed Size",
#     color="red",
# )
# ax2.bar(
#     bins_nf[:-1],
#     norm_counts_nf,
```

```
#     width=np.diff(bins_nf),
#     alpha=1,
#     label="Spanning Integer",
#     color="black",
# )


ax1.set_yscale("log")
# ax1.set_xscale("log")
ax1.set_ylim(1e-4, 5e-1)
ax1.set_xlim(0, 16)
# ax1.set_xscale("log")
# ax1.set_title("Packing Set Convergence")

ax2.set_yscale("log")
# ax1.set_xscale("log")
ax2.set_ylim(5e-18, 5e-16)

ax2.set_xlabel(r"Iterations (and q$_{nS}$)")
ax1.set_ylabel("Packing Set Error")

arrowprops = dict(arrowstyle="->", color="k", lw=1)
bbox = dict(fc="1",ec="1")

ax1.annotate(
    r"$\epsilon_{tol}$",
    xy=(10, tol),
    xytext=(12, 1e-3),
    fontsize=14,
    arrowprops=arrowprops,
    bbox=bbox,
    color="k",
    ha="left",
    va="center",
)

ax2.annotate(
    "If any iteration needed,\nSI converges to numeric zero quickly.",
    xy=(2.5, 1e-16),
    xytext=(3.5, 5e-17),
    fontsize=12,
    arrowprops=arrowprops,
```

```
    bbox=bbox,
    color="k",
    ha="left",
    va="center",
)

from matplotlib.lines import Line2D

custom_lines = [
    Line2D([0], [0], color="red", lw=2, alpha=0.5, label="Fixed Size"),
    Line2D([0], [0], color="black", lw=2, alpha=0.5, label="Spanning Integer"),
]

# fig, ax = plt.subplots()
# lines = ax.plot(data)
ax1.legend(title="MPS Algorithm", handles=custom_lines, loc='upper right')
```

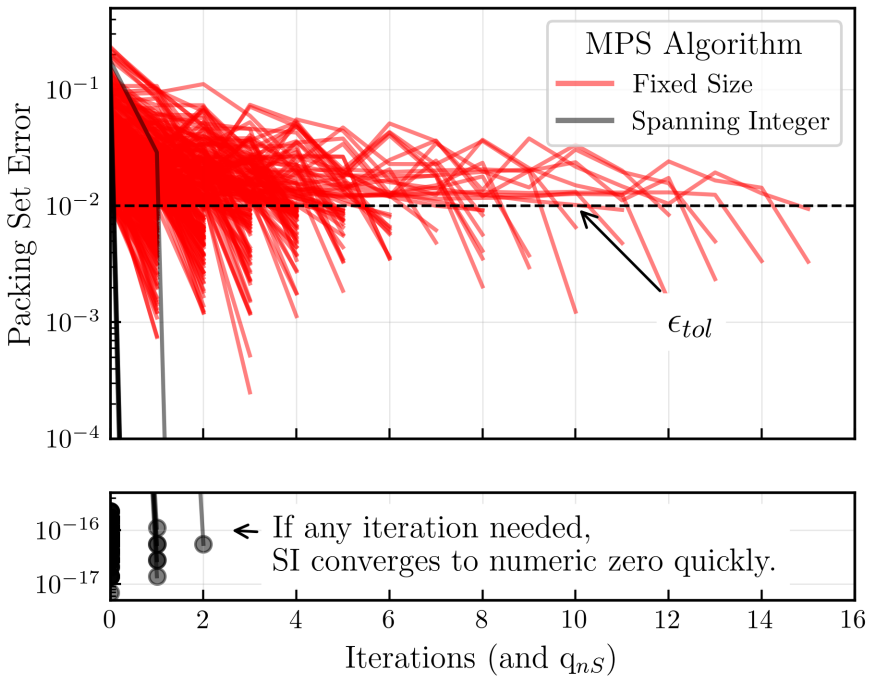

Figure 2: Packing Algorithm Convergence

Note that this solution for the minimal packing set is dependent on the selected value for $x_{n,S}$. Because the maximum particle size, $x_{n,S}$ controls the packing set, and a smaller value for $x_{n,S}$

will tend to minimize the set, if the full range of possible $x_{n,S}$ is considered, $x_{n,S}$ and $x_{n-1,S}$ will tend to converge. The result of that convergence would be equivalent to the solution for a target G with $x_{n-1,S} \approxeq x_{n,S} = x_{n-1,G}$ and the $x_{n-1,G}$ sieve removed.

Grain size effects on MPS

Regarding $||S_{mp}||$, it tends to be controlled by two factors: the maximum value in Y and the value in the last entry in Y. In general terms, the first is a function of the ratio of the largest to smallest particles in the sample. The second is a function of the curvature of the sample.

```
# Create a larger dataset for plotting trends in MPS for a broad spectrum of GSDs
## Set up data structure to store GSDs and MPSs
def add_data_row(data_rows, mpgen: MinimalPackingGenerator):
    """
    Adds a row to the DataFrame with the provided keyword arguments.
    Any missing columns will be filled with NaN.

    """
    sample = mpgen.mps
    total_particles = sum(sample.quantities)
    mass_max = sample.total_masses[-1]
    mass_mid = sample.total_masses[len(sample.total_masses) // 2]
    mass_min = sample.total_masses[0]
    mass_ratio = mass_min / mass_max
    percent_fines = mass_min / sum(sample.total_masses)

    size_max = sample.sizes[-1]
    size_min = sample.sizes[0]
    size_ratio = int(np.round(size_max / size_min, 0))

    row = {
        "GSD": mpgen.g,
        "n_sieves": len(sample.sizes),
        "mass_ratio": mass_ratio,
        "size_ratio": size_ratio,
        "vol_ratio": size_ratio**3,
        "percent_fines": percent_fines,
        "total_particles": total_particles,
        "d_10": mpgen.g.d_10,
        "d_30": mpgen.g.d_30,
        "d_60": mpgen.g.d_60,
        "cc": np.round(mpgen.g.cc, 3),
```

```
        "cu": np.round(mpgen.g.cu, 3),
        "gs_index": mpgen.g.gs_index,
        "curvature_index": mpgen.g.curvature_index,
        "log_size": mpgen.g._i_gs_curve()[0],
        "gsd_curve": mpgen.g._i_gs_curve()[1],
        "slope": mpgen.g.slope,
        "curvature": np.mean(mpgen.g.curvature),  # + size_ratio**3,
        # Best so far:
        "shape_factor": np.log10(
            size_ratio**3 * (1 + np.mean(mpgen.g.curvature))
        ),  # * np.log10(size_ratio)**2),
        "asdf": mass_ratio * size_ratio**3,
        "group_symbol": mpgen.g.uscs_symbol,
        "group_name": mpgen.g.uscs_name,
    }
    data_rows.append(row)
    pass

## Populate dataset
data_rows = []
x = x1
min_span = 5
for start in range(0, len(x) - min_span):
    for end in range(start + min_span, len(x)):
        new_x = x[start:end]
        n_sieves = len(new_x)

        for j in range(50):
            # Create a distributed set of retained masses
            mass = mass_dist(n_sieves, rng, exponent=2, extra_randomness=7)

            mass[-1] = 0.0  # Ensure last mass is zero
            g = GSD(sizes=new_x, masses=mass)

            # Create a flexible minimal packing for each GSD
            mps = MinimalPackingGenerator(
                g, x_n_factor=0.5, tol=tol, flex=True, density=1.0
            )
            add_data_row(data_rows, mps)

## Convert the list of dictionaries to a DataFrame
df = pd.DataFrame(data_rows)
```

```
/var/folders/q0/kxmqm5c95n7cxc6mmqklw1k40000gq/T/ipykernel_65436/2294212876.py:41: RuntimeWa
  "shape_factor": np.log10(
```

Implications for DEM modeling

The algorithms described above can 1. Find a first-order approximation of the minimal packing. 2. Quantify the error in approximate minimal packings. 3. Find a rigorous solution minimal packing.

```
color_param = "vol_ratio"
x_param = "mass_ratio"  # "curvature_index"
y_param = "total_particles"

fig, ax = plt.subplots()
ax.scatter(
    df[x_param],
    df[y_param],
    c=np.log10(df[color_param]),
    s=10,
    cmap="viridis",
    alpha=1,
)

# ax.legend(title="cu", loc='upper left', bbox_to_anchor=(1, 1))
colorbar = plt.colorbar(ax.collections[0], ax=ax)
colorbar.set_label(r"Log$_{10}$Volume Ratio ($\Xi_1$)")

ax.set_xlabel(r"Mass Ratio ($\Phi_1$)")
ax.set_ylabel("Total Particles in Packing Set")
ax.set_xscale("log")
ax.set_yscale("log")
```

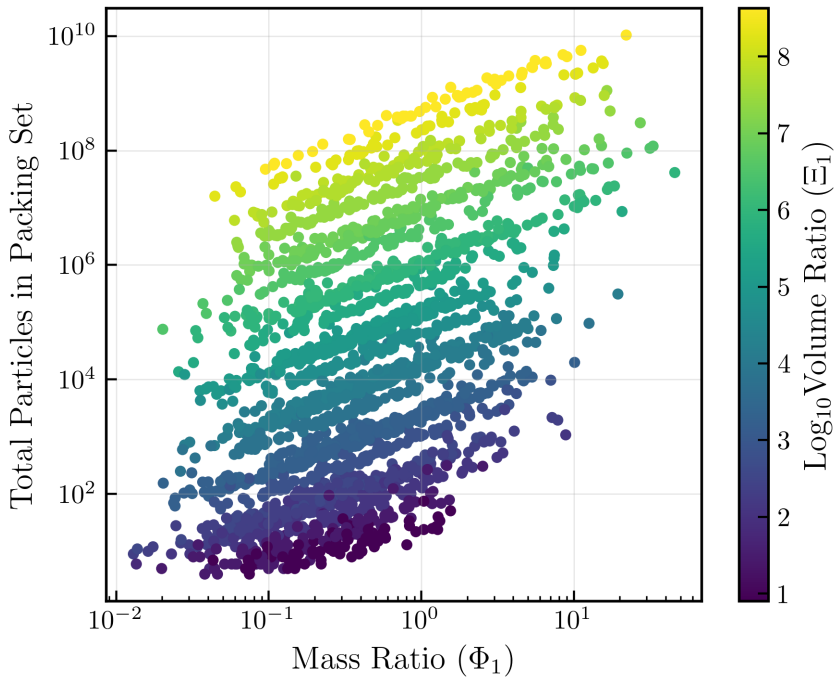

Figure 3: Change in total number of MPS Particles with mass ratio.

```
color_param = "vol_ratio"
x_param = "curvature_index"
y_param = "total_particles"

fig, ax = plt.subplots()
ax.scatter(
    df[x_param],
    df[y_param],
    c=np.log10(df[color_param]),
    s=10,
    cmap="viridis",
    alpha=1,
)

# ax.legend(title="cu", loc='upper left', bbox_to_anchor=(1, 1))
colorbar = plt.colorbar(ax.collections[0], ax=ax)
colorbar.set_label(r"Log$_{10}$Volume Ratio ($\Xi_1$)")

ax.set_xlabel("Curvature Index")
ax.set_ylabel("Total Particles in Packing Set")
```

```
# ax.set_xscale("log")
ax.set_yscale("log")
```

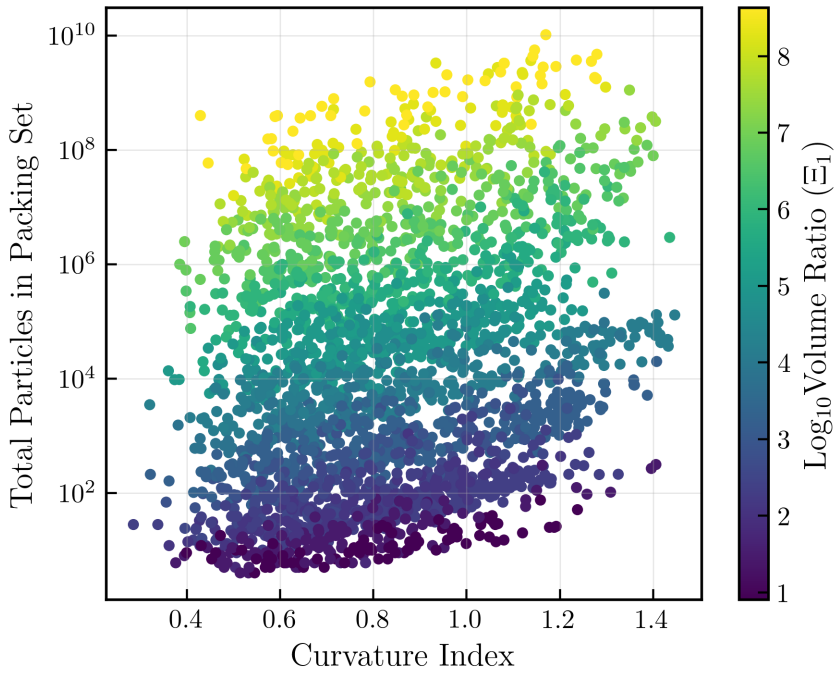

Figure 4: Change in total number of MPS Particles with curvature index.

```
fig, ax = plt.subplots(figsize=(6,5),layout="constrained")

x_param = "gs_index"
y_param = "total_particles"
group_param = "group_name"
# df_plot = df[df[group_param].str.contains("")]

categories = ["graded gravel", "graded sand", "silty"]
markers = ["1", ".", "."]

for j, cat in enumerate(categories):
    df_plot = df[df[group_param].str.contains(cat)]


    if cat == "silty":
        palette = ["#77B3C9", "#F2A900"]
```

```
    else:
        palette = sns.color_palette()  # "Set2", len(df_plot[group_param].unique()))

    for i, (label, group) in enumerate(df_plot.groupby(group_param)):
        # if "graded gravel" in label:
        #     marker = "|"
        # else:
        #     marker = "_"
        ax.scatter(
            group[x_param],
            group[y_param],
            label=f"{label}",
            marker=markers[j],
            color=palette[i],
            # alpha=0.75,
            s=40,  # Scale point size by log_total_particles
        )


fig.legend(
    loc="outside upper center",
    fontsize="small",
    ncols=2,
    frameon=False,
    handlelength=2,
)

ax.set_xlabel("Grain Size Index")
ax.set_ylabel("Total Particles in Packing Set")
ax.set_ylim(1e0,2e10)
ax.set_xlim(0.0, 0.7)
# ax.set_xlim(1e-2, 1e2)
# ax.set_xscale("log")
ax.set_yscale("log")
ax.grid(True, which='both', axis='both')
```

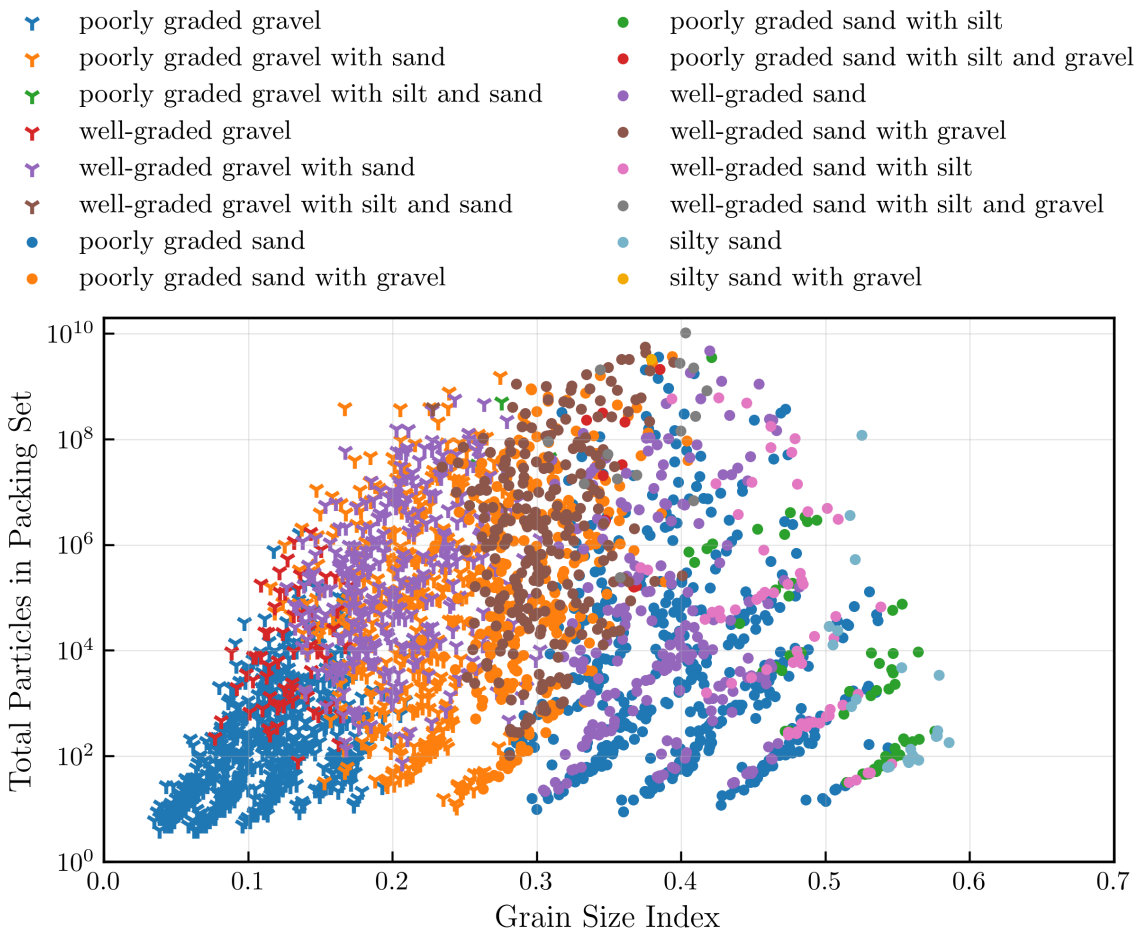

Figure 5: USCS classifications of the MPS data.

The rigorous minimal packing may be significantly larger than the first-order approximation or some other packing set within some acceptable tolerance. Whether a rigorous or approximate solution is needed depends very much on the application. But these concepts can be used to efficiently quantify the computational cost of rigor in the $S : G$ match.

Insert a plot showing a realistic GSD and the sieve-specific error trend with increasing N. I could draw a typical GSD with a dual axis with N on the right. There would be several size-N lines with a colorscale indicating their error? or I could plot several GSDs and color by N and use the other axis for the error.

They can also be used to evaluate the computational cost of approximate and rigorous minimal packings for different USCS classifications.

Insert a plot showing error v. N for granular USCS soils.

Cundall, P. A. 2001. "A Discontinuous Future for Numerical Modelling in Geomechanics?" *Proceedings of the Institution of Civil Engineers - Geotechnical Engineering* 149 (1): 41–47. https://doi.org/10.1680/geng.2001.149.1.41.

Cundall, P. A., and O. D. L. Strack. 1979. "A Discrete Numerical Model for Granular Assemblies." *Géotechnique* 29 (1): 47–65. https://doi.org/10.1680/geot.1979.29.1.47.

D18 Committee. n.d. "Test Methods for Particle-Size Distribution (Gradation) of Soils Using Sieve Analysis." https://doi.org/10.1520/d6913_d6913m-17.

Dong, Youkou, Dingtao Yan, and Lan Cui. 2022. "An Efficient Parallel Framework for the Discrete Element Method Using GPU." *Applied Sciences* 12 (6): 3107. https://doi.org/10.3390/app12063107.

Fang, Luning, Ruochun Zhang, Colin Vanden Heuvel, Radu Serban, and Dan Negrut. 2021. "Chrono::GPU: An Open-Source Simulation Package for Granular Dynamics Using the Discrete Element Method." *Processes* 9 (10): 1813. https://doi.org/10.3390/pr9101813.

O'Sullivan, C. 2014. "Advancing Geomechanics Using DEM." In, edited by Kenichi Soga, Krishna Kumar, Giovanna Biscontin, and Matthew Kuo, 21–32. CRC Press. http://www.crcnetbase.com/doi/abs/10.1201/b17395-4.

Sufian, Adnan, Marion Artigaut, Thomas Shire, and Catherine O'Sullivan. 2021. "Influence of Fabric on Stress Distribution in Gap-Graded Soil." *Journal of Geotechnical and Geoenvironmental Engineering* 147 (5): 04021016. https://doi.org/10.1061/(ASCE)GT.1943-5606.0002487.

Zhang, Ruochun, Bonaventura Tagliafierro, Colin Vanden Heuvel, Shlok Sabarwal, Luning Bakke, Yulong Yue, Xin Wei, Radu Serban, and Dan Negruţ. 2024. "Chrono DEM-Engine: A Discrete Element Method Dual-GPU Simulator with Customizable Contact Forces and Element Shape." *Computer Physics Communications* 300 (July): 109196. https://doi.org/10.1016/j.cpc.2024.109196.